Evaluation of Some Blockcipher Modes of Operation

Phillip Rogaway
University of California, Davis
Dept. of Computer Science
Davis, California, USA

E-mail: rogaway@cs.ucdavis.edu
URL: http://www.cs.ucdavis.edu/∼rogaway

February 10, 2011

Evaluation carried out for the
Cryptography Research and Evaluation Committees (CRYPTREC)
for the Government of Japan

Contents

Acknowledgments

Many thanks to **Mihir Bellare** for his drafting the chapter on HMAC. We also corresponded on other random matters that came up as I carried out this study. More broadly, many of the viewpoints embodied in this evaluation were co-developed with Mihir over a great many years.

I received numerous insightful and useful comments, corrections, and answers to questions from colleagues **Morris Dworkin**, **Niels Ferguson**, **Shai Halevi**, **Viet Tung Hoang**, **Ted Krovetz**, **David McGrew**, **Chanathip Namprempre**, **Bart Preneel**, and **Kan Yasuda**. My heartfelt thanks to everyone named for all your time and kind assistance.

The work of this report was supported by the Cryptography Research and Evaluation Committees (CRYPTREC), Japan. My contacts at the Mitsubishi Research Institute have included **Dai Mochinaga**, **Miyako Ohkubo**, and **Sachiko Yamada**. Thanks for arranging contract formalities, answering questions, and providing documentation. I hope my report will serve your needs well.

Phillip Rogaway

February 2011

Chapter 1

Summary

1.1. Overview. This report analyzes the security of some 17 cryptographic modes of operation described within some eight U.S. or international standards. Most of the schemes are well-known; many are widely used. The modes under consideration are the encryption schemes ECB, CBC, CFB, OFB, CTR, and XTS; the message authentication codes CMAC, HMAC, GMAC, and MAC Algorithms 1–6 of ISO 9797-1:1999; and the authenticated-encryption schemes CCM and GCM. The containing standards are FIPS 198-1, ISO/IEC 9797-1:1999, NIST SP 800-38A, NIST SP 800-38B, NIST SP 800-38C, NIST SP 800-38D, NIST SP 800-38E, and by reference from the last standard, IEEE 1619-2007 [61–65, 90, 91, 159].

Despite the modes being standardized and well-known, the quality varies. Some schemes are quite sensible and modern, while the value of others seems to be mostly in their legacy significance, or as building blocks for other schemes.

In many cases it is unclear, to me, if a mode "ought" to be included in the CRYPTREC portfolio; the problem is that some well-entrenched schemes are, in fact, rather poor and dated designs. Correspondingly, I take my main goal as the description of what is known about each scheme, rather than an explication of what I think "should" be done. Still, I sometimes do offer opinions.

I have tried to avoid being overly technical in these pages. The scope is too large, and the schemes too well-studied, for it to make sense to try to write up fresh proofs for everything. Doing so would easily turn this already-long manuscript into a book-length treatment. Instead, I have tried to explain the various results, point the reader to the relevant literature, and explain how, I think, the results should be interpreted.

I divide the modes into three categories: (I) confidentiality modes, (II) authenticity modes, and (III) authenticated-encryption modes. See Figure 1.1. When I contracted for this project, CRYPTREC organized matters differently: eight techniques partitioned into two categories, named "modes of operation" and "message authentication codes." I would like to clarify that *all* the schemes of this report can be viewed as *modes of operation*.

1.2. Evaluative approach. I have tried to instill a degree of uniformity in the evaluative process. I will, for each mode, be answering some or all of the following questions

1.2.1 What **cryptographic problem** does the mode try to solve? Problem identification is crucial, yet often ignored. A definition is sought, in the tradition of provable-security cryptography. Sometimes problem identification is trivial; we know, for example, that CCM is supposed to be a nonce-based authenticated-encryption scheme because the

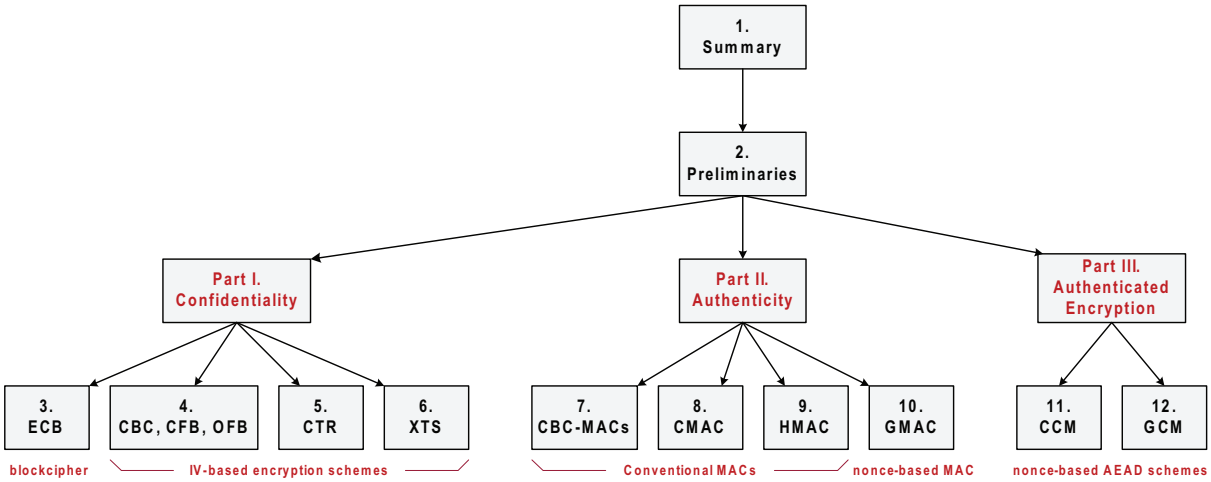


Figure 1.1: **Roadmap**. The chart shows organization and logical dependencies among the chapters and parts of this documents.

designers *said* this, and because the mode springs from that tradition. But what is the cryptographic problem a mode like ECB or XTS is supposed to solve? This is a less trivial question, and sometimes one that cannot be as definitively answered. Modes are often designed to achieve many aims, not all of them clear, formalized, or well understood.

1.2.2 What does the apparatus of **provable security** have to say about the mode's security? Can one establish that the mode does its now-identified job under well-believed cryptographic assumptions? If so, under *what* assumptions? For blockcipher-based schemes, the preferred assumption is security as a pseudorandom-permutation (PRP). How *tight* are the reductions to breaking the underlying PRP?

1.2.3 What **attacks** are known against the scheme? Is there a quantitative gap between the known attacks and the proven bounds? Does the gap matter?

1.2.4 How **efficient** is the scheme? There are multiple characteristics that can matter for efficiency, and the relevant ones depend on that scheme's goal.

1.2.5 How **widely-used** is the scheme already? If a mode has a long history or is extensively deployed, this alone can be a reason for standardization, other problems with the scheme notwithstanding.

1.2.6 How **simple** is the scheme? Good modes of operation are pretty things, elegant and minimal for accomplishing their aims.

1.2.7 How **robust** is the scheme against misuse? If one can expect that a scheme will routinely be misused—used in ways contrary to what is required of the mode or guaranteed by the mode—this is certainly a problem.

1.2.8 How **well understood** is the scheme? Has the mechanism been widely studied? Have the important questions been answered, or do there remain important gaps in what we know?

1.2.9 How good is the **specification document** that describes the mode? While some might claim that a mechanism transcends the virtues or failings of its description, I believe the opposite, that the quality of the specification is part and parcel of the quality of the

scheme. For one thing, the specification document impacts the likelihood of a scheme being correctly or incorrectly used.

Another aspect of this report is the simple fact of **organizing** this menagerie of modes in some coherent way. The taxonomy of Figure 1.1 was not the only possible approach.

1.3. The positive role of the standards bodies. This report takes a fresh look at eight different standards—six from NIST and one each from the IEEE and ISO/IEC. Overall, the assessment I will give may sound fairly critical about this body of work. This does not represent my actual view. Quite the opposite; the modes are, collectively, quite good, and NIST, in particular, has shown commendable leadership in their work on standardizing (or "recommending") modes of operation. If it was once true that, in cryptography, each standards body liked to wait around for the other to act, this most definitely is *not* the case today.

Some of the negativism one will see in this report may be a bit half-hearted and *pro forma*: academics are *supposed* to be critical of what we review; it is wired in our brains. Any piece of completed work *could* have been better done, and a critique *should* bring out these shortcomings. A negative-sounding critique should not be understood as an overall negative opinion of a standard or a mode contained therein; it is par for the course.

More concretely, I would make the following comments to help balance the possibly negative-sounding tenor of this report. First, that *all* of the modes embodied in FIPS 198-1 and NIST Recommendations SP 800-38B (CMAC), SP 800-38C (CCM), SP 800-38D (GCM), and SP 800-38E—standards for HMAC, CMAC, CCM, GCM, and XTS, respectively—owe their *existence* to the provable-security tradition. In standardizing this set of techniques, NIST has ushered in a new age in symmetric cryptography, one where everything "above" the level of a blockcipher (or hash function or compression function) is designed using, and proven with, the provable-security chest of tools and ideas. Recommendation SP 800-38A, while not standardizing any fundamentally new technique, expanded the repertoire of sanctioned modes by including a method—CTR mode—that gains its importance equally from the current nature of processor and hardware design *and* from our current understanding of provable-security cryptography. The standardization of the named modes has a strongly positive impact in bringing about safer shared-key encryption, authenticated encryption, and message authentication.

Whatever criticism I may raise about this method or that, or about some phrase, paragraph, appendix, idea, or result, the overall context and summary opinion should not be lost: **(a)** that having standards for symmetric cryptography—standards reaching above the level of blockciphers—is important and desirable; **(b)** that these standards should be well-informed by the science that the cryptographic community has been working to create; **(c)** and that this is precisely the direction we are seeing, overall, from the set of standards I was invited to review.

There is a reason that standards often take years to complete. The reason is not politics, bickering, or stupidity, as I have heard claimed more than once. The reason is that fashioning a good standard is *hard*. It is both hard technically and hard because a standard that has value is an artifact that, almost always, can only emerge after a high degree of dialog arising within a particular socio-technical institution. I have much respect and appreciation for those who have worked hard to bring us the modes reviewed in these pages. I hope that those who have had a hand in creating these modes and standards will take my comments as quiet and constructive suggestions and ideas.

1.4. Some "types" of cryptographic objects. We now introduce some terms we will be using for the remainder of this chapter. Later in this report, we will be describing all of these

terms in greater depth; still, we must introduce the terms now for our summary to make sense.

When we speak of an **IV-based encryption scheme** we mean that the "syntax" of the mechanism looks like $C = \mathcal{E}_K^{IV}(P)$ where K is a key, P is a plaintext, C is a ciphertext, and IV is an "initialization vector." We are not speaking of the scheme's security characteristics. A **probabilistic encryption scheme** $C = \mathcal{E}_K^R(P)$ is an IV-based encryption scheme, syntactically, but we are suggesting that, in the security definition, the IV will be regarded as a random value R. A **nonce-based encryption scheme** $C = \mathcal{E}_K^N(P)$ is an IV-based encryption scheme, syntactically, but we are suggesting that, in the security definition, the IV will be regarded as a nonce N: a value that is used at most *once* for a given key. When we speak of a **blockcipher** $C = \mathcal{E}_K(P)$ we mean a keyed, length-preserving permutation. We are not suggesting any particular security property. A **message authentication code** (MAC) $T = F_K(M)$ is a keyed function of strings that returns a fixed-length tag. The notion of security will be unforgeability under an adaptive chosen-message attack. A **nonce-based MAC** $T = F_K^N(M)$ is a like a MAC with an additional argument, a nonce N. Finally, when we speak of a **nonce-based AEAD scheme** (authenticated-encryption with associated-data) $C = \mathcal{E}_K^{N,A}(P)$ we mean a nonce-based encryption scheme with one extra argument, the *associated data*, a string A. The intent is that C protects the confidentiality *and* authenticity of P, as well as the authenticity of A.

1.5. Confidentiality modes. Summary findings on six different confidentiality modes are given in Figure 1.2. The first four modes (the "basic four") are classical, going back more than 30 years, to FIPS 81 [153] (at which time the modes were specific to DES). Yet none of these are terribly sensible schemes from a modern point of view; ECB doesn't come close to receiving the classically-understood "goal" of a confidentiality scheme—semantic security under a chosen-plaintext (CPA) attack, or notions equivalent to it [14, 74]—while other modes achieve it, assuming a random IV, but at a pointlessly high cost. Most of these schemes were invented to address issues like error-propagation, or an attempt to deny the adversary chosen plaintext/ciphertext pairs, goals that arguably have almost *no* significance from the point of view of how we currently understand cryptography or, for that matter, good security architecture. Re-standardization of some or all of the basic-four modes would be primarily justified by legacy considerations. In the cases of ECB and CBC, the modes are also relevant as building blocks for creating *other* blockcipher modes.

The next mode, CTR, was apparently suggested by Diffie and Hellman just as early as the basic-four modes [58], yet for some reason it was not included in the initial batch. I regard CTR as the "best" choice among the classical confidentiality-only techniques. Its parallelizability and obvious correctness, when based on a good blockcipher, mean that the mode should be included in any modern portfolio of modes.

The last scheme, XTS, is a relative newcomer; it was only standardized by NIST in January 2010 [65], following IEEE standardization in 2007 [90]. Unlike the other modes reviewed in this document, XTS is intended for a specific application: encrypting fixed-length "data units" (eg, disk sectors) on a storage device. Whereas ECB leaks the equality of blocks across time and position, XTS leaks the equality of blocks across time but not position. The mode is inherently a compromise, achieving weaker goals than what a tweakable blockcipher "should" achieve if it is applied to a possibly-long string, but running more efficiently.

1.6. Authenticity modes. Summary findings for the reviewed authentication modes are in Figure 1.3. I begin by looking at the cornucopia of MACs that were included in ISO 9797-1. By one natural reckoning, that standard encompasses some *twenty-one* different schemes. The request from CRYPTREC was to review the CBC-MAC from this standard—but they're *all*

Mode from	**Summary evaluation**
ECB SP 800-38A	A **blockcipher**, the mode enciphers messages that are a multiple of n bits by separately enciphering each n-bit piece. The security properties are weak, the method leaking equality of blocks across both block positions and time. Of considerable legacy value, and of value as a building block for other schemes, but the mode does not achieve any generally desirable security goal in its own right and must be used with considerable caution; ECB should not be regarded as a "general-purpose" confidentiality mode.
CBC SP 800-38A	An **IV-based encryption scheme**, the mode is secure as a probabilistic encryption scheme, achieving indistinguishability from random bits, assuming a random IV. Confidentiality is not achieved if the IV is merely a nonce, nor if it is a nonce enciphered under the same key used by the scheme, as the standard incorrectly suggests to do. Ciphertexts are highly malleable. No chosen-ciphertext attack (CCA) security. Confidentiality is forfeit in the presence of a correct-padding oracle for many padding methods. Encryption inefficient from being inherently serial. Widely used, the mode's privacy-only security properties result in frequent misuse. Can be used as a building block for CBC-MAC algorithms. I can identify no important advantages over CTR mode.
CFB SP 800-38A	An **IV-based encryption scheme**, the mode is secure as a probabilistic encryption scheme, achieving indistinguishability from random bits, assuming a random IV. Confidentiality is not achieved if the IV is predictable, nor if it is made by a nonce enciphered under the same key used by the scheme, as the standard incorrectly suggests to do. Ciphertexts are malleable. No CCA-security. Encryption inefficient from being inherently serial. Scheme depends on a parameter s, $1 \leq s \leq n$, typically $s = 1$ or $s = 8$. Inefficient for needing one blockcipher call to process only s bits . The mode achieves an interesting "self-synchronization" property; insertion or deletion of any number of s-bit characters into the ciphertext only temporarily disrupts correct decryption.
OFB SP 800-38A	An **IV-based encryption scheme**, the mode is secure as a probabilistic encryption scheme, achieving indistinguishability from random bits, assuming a random IV. Confidentiality is not achieved if the IV is a nonce, although a fixed sequence of IVs (eg, a counter) does work fine. Ciphertexts are highly malleable. No CCA security. Encryption and decryption inefficient from being inherently serial. Natively encrypts strings of any bit length (no padding needed). I can identify no important advantages over CTR mode.
CTR SP 800-38A	An **IV-based encryption scheme**, the mode achieves indistinguishability from random bits assuming a nonce IV. As a secure nonce-based scheme, the mode can also be used as a probabilistic encryption scheme, with a random IV. Complete failure of privacy if a nonce gets reused on encryption or decryption. The parallelizability of the mode often makes it faster, in some settings much faster, than other confidentiality modes. An important building block for authenticated-encryption schemes. Overall, usually the best and most modern way to achieve privacy-only encryption.
XTS SP 800-38E & IEEE 1619	An **IV-based encryption scheme**, the mode works by applying a tweakable blockcipher (secure as a strong-PRP) to each n-bit chunk. For messages with lengths not divisible by n, the last two blocks are treated specially. The only allowed use of the mode is for encrypting data on a block-structured storage device. The narrow width of the underlying PRP and the poor treatment of fractional final blocks are problems. More efficient but less desirable than a (wide-block) PRP-secure blockcipher would be.

Figure 1.2: **Summary findings — confidentiality modes.** The list includes the four "classical" modes (ECB, CBC, CFB, OFB), a parallelizable (no chaining) alternative (CTR), and a recent scheme for on storage-device encryption (XTS).

Mode from	Summary evaluation
ALG 1–6 ISO 9797-1: 1999	A **collection of MACs**, all of them based on the CBC-MAC. Too many schemes. Some are provably secure as VIL PRFs, some as FIL PRFs, and some have no provable security. Some of the schemes admit damaging attacks. Some of the modes are dated. Key-separation is inadequately attended to for the modes that have it. Should not be adopted *en masse*, but selectively choosing the "best" schemes is possible. It would also be fine to adopt none of these modes, in favor of CMAC. Some of the ISO 9797-1 MACs are widely standardized and used, especially in banking. A revised version of the standard (ISO/IEC FDIS 9797-1:2010) will soon be released [93].
CMAC SP 800-38B	A **MAC** based on the CBC-MAC, the mode is provably secure (up to the birthday bound) as a (VIL) PRF (assuming the underlying blockcipher is a good PRP). Essentially minimal overhead for a CBCMAC-based scheme. Inherently serial nature a problem in some application domains, and use with a 64-bit blockcipher would necessitate occasional re-keying. Cleaner than the ISO 9797-1 collection of MACs.
HMAC FIPS 198-1	A **MAC** based on a cryptographic hash function rather than a blockcipher (although most cryptographic hash functions are themselves based on blockciphers). Mechanism enjoys strong provable-security bounds, albeit not from preferred assumptions. Multiple closely-related variants in the literature complicate gaining an understanding of what is known. No damaging attacks have ever been suggested. Widely standardized and used.
GMAC SP 800-38D	A **nonce-based MAC** that is a special case of GCM. Inherits many of the good and bad characteristics of GCM. But nonce-requirement is unnecessary for a MAC, and here it buys little benefit. Practical attacks if tags are truncated to ≤ 64 bits and extent of decryption is not monitored and curtailed. Complete failure on nonce-reuse. Use is implicit anyway if GCM is adopted. Not recommended for separate standardization.

Figure 1.3: **Summary findings — authenticity modes.** The various authenticity modes (MACs) reviewed in this report. All but GMAC are pseudorandom functions (PRFs) as well as MACs.

CBC-MACs, under a sufficiently catholic understanding of the word, and I decided I ought to consider them all. The chapter on the ISO 9797-1 scheme—not surprisingly, the longest of the lot—tries to go through what is and isn't known about all the different CBC-MAC variants. Overall, I am rather critical of this standard, which I see as fundamentally too open-ended, and particularly sloppy when it comes to key-separation. I do not recommend blanket adoption of ISO 9797-1 schemes. A new version of ISO/IEC 9797-1 will soon be forthcoming; a revised Final Draft International Standard (FDIS), document ISO/IEC FDIS 9797-1:2010, has just been approved [93]. While not in scope of this review (and while located too late to be reviewed, regardless), it appears that the revised 9797-1 draft addresses many, but not all, of the concerns I raise regarding the ISO/IEC 9797-1:1999 standard.

Next I look at CMAC and HMAC, both of which I like. They are simple and modern schemes, with good provable security results. The second mode, especially, is by now quite widely deployed.

Finally, I consider GMAC, which is a special case of GCM (reviewed in the authenticated-encryption part of this report). I find significant problems with the mode: it does not work well when tags are substantially truncated, and the absolute necessity of supplying a (never-repeating) nonce means that the scheme is needlessly vulnerable (for a MAC) to misuse. I would advise against separately standardizing this scheme, which would, in any case, be "automatically" covered by any standard covering GCM.

1.7. Authenticated-encryption modes. Summary findings for two authenticated-encryption

Mode from	Summary evaluation
CCM SP 800-38C	A **nonce-based AEAD scheme** that combines CTR mode encryption and the raw CBC-MAC. Inherently serial, limiting speed in some contexts. Provably secure, with good bounds, assuming the underlying blockcipher is a good PRP. Ungainly construction that demonstrably does the job. Simpler to implement than GCM. Can be used as a nonce-based MAC. Widely standardized and used.
GCM SP 800-38D	A **nonce-based AEAD scheme** that combines CTR mode encryption and a $\mathrm{GF}(2^{128})$-based universal hash function. Good efficiency characteristics for some implementation environments. Good provably-secure results assuming minimal tag truncation. Attacks and poor provable-security bounds in the presence of substantial tag truncation. Can be used as a nonce-based MAC, which is then called GMAC. Questionable choice to allow nonces other than 96-bits. Recommend restricting nonces to 96-bits and tags to at least 96 bits. Widely standardized and used.

Figure 1.4: **Summary findings — authenticated-encryption modes.** Here we summarize findings for the two authenticated-encryption modes reviewed in this report. Both are nonce-based and can process incoming associated data, such as a message header.

schemes, CCM and GCM, are in Figure 1.4. Both are nonce-based (meaning that the user must supply a never-repeating IV). Both allow "associated-data" (for example, a message header) to be authenticated alongside whatever is being encrypted. The methods are therefore sometimes termed schemes for AEAD—authenticated-encryption with associated-data. I offer substantial criticism on both CCM and GCM, but none of the criticisms are particularly new. For CCM, the serial nature of the scheme, for both encryption and decryption, is unpleasant, as is its failure to be on-line. The latter aspect, especially, is fundamentally unnecessary. For GCM, we complain most stridently about security if tag lengths are short (where even 64 bits must be considered short, given the security results). Still, in the end, I must come down in favor of including both CCM ad GCM in any portfolio of modern AEAD schemes: the mechanisms are widely used, and can be used safely. The overall migration from pure-privacy modes (CBC and its friends) to AEAD modes (CCM, GCM, and the like) represents an important shift in the "service" that users can expect symmetric encryption schemes to provide, a shift that is likely to result in decreasing misuse.

1.8. Organization. The evaluation of each mode has been dropped into its own chapter with the exception of three IV-based encryption schemes—CBC, CFB, and OFB—which I treat together, and the various CBC-MAC modes of ISO 9797-1, which I likewise lump together. Background material for each of the three parts of this report is given the "pseudo-chapters" that are labeled as Parts I–III. Finally, there is, after this summary, a chapter of preliminaries that contains material useful for the overall enterprise. While there are some synergies to be found by reading everything, I have tried to keep the chapters independent of one another. A partial exception is the chapter on GMAC, which might benefit from a prior reading of the chapter on GCM. See again the chart of Figure 1.1.

1.9. New material. I would identify the following aspects of this report as at least somewhat novel, going beyond what I, at least, had understood to be the current state of thinking on modes of operation.

1.9.1 Our (syntactic) formulation for IV-based encryption schemes is new, as is the viewpoint we select that makes probabilistic and nonce-based security different notions levied on the same underlying object. An advantage of this approach is that it lets one formally

claim that nonce-based security is properly stronger than probabilistic security (before, such a statement could only be understood as a "meta" assertion, since, formally, nonce-based and probabilistic encryption schemes were different kinds of objects). I also employ a syntax for IV-based encryption schemes that follows the traditional approach for defining blockciphers—that is, just a single algorithm, rather than a three-tuple or the like. I go on to use the single-algorithm formulation for AEAD schemes too, a nice syntactic simplification for these objects as well.

1.9.2 I describe a formalization for the (limited) privacy goal to which ECB and XTS apparently aim. A "filter function" $\mathcal{F}$ can be defined such that one can speak of ECB being private up to the leakage of $\mathcal{F}_{\text{ECB}}$. Similarly, one can speak of being private up to the leakage of $\mathcal{F}_{\text{XTS}}$, or private up to leaking *any* piece of information $\mathcal{F}(C)$ about ciphertexts. I suggest adapting this approach to similarly speak of nonmalleability up to some function $\mathcal{F}$, so that one could speak of just "how malleable" a scheme like XTS is.

1.9.3 I suggest that Appendix C of NIST SP 800-38A is wrong to recommend that, to create the IV for CBC or CFB modes, one can "apply the forward cipher function, under the same key that is used for encryption of the plaintext, to a nonce" [61, p. 20]: the resulting scheme does not achieve what we define as the intended goal. The spec is similarly a bit off in suggesting that the IVs for OFB need only be nonces; while unpredictability is not needed, there are attacks if the sequence of distinct IVs—nonces—may be adaptively changed.

1.9.4 Our formalization for CTR mode is new. At issue is the fact that SP 800-38A doesn't indicate *how* counters are to be formed and so, to match this spec, it does not really work to assume some specific counter-increment technique, as was done by Bellare, Desai, Jokipii, and Rogaway [14].

1.9.5 I try to clarify that there are effectively three tweakable blockciphers associated to the definition of XTS, and we reformulate the mode in a way that helps clarify its structure and make it more transparent how XEX "should" have been enlarged if one aimed to do well enciphering substrings of $\eta \geq n$ bits.

1.9.6 I explain that, despite there being proofs in the standard model for many of the ISO 9797-1 CBCMAC-variants [91], that it would still be good to have good PRF-advantage bounds for ideal-cipher-model. By explicitly attending to key length, such bounds would help one to assess the efficacy of attacks that run in a certain amount of time and use a certain number of message/MAC pairs. I suspect that cryptographers have too quickly dismissed the utility of ideal-cipher-model proofs in this domain because standard-model proofs were available and assumed preferable, rather than incomparable. With respect to HMAC and other iterated MACs, we identify as an important open problem to *prove* that a limited amount of truncation can improve security beyond the birthday-bound.

1.9.7 I point out that Ferguson's attack [67] on GCM applies equally well to GMAC. I explain the serious shortcoming of the bounds asserted by McGrew and Viega [133] in the face of substantial tag-truncation. It seems not to have been clearly acknowledged by the authors or others that the bounds for GCM are, in the presence of shortened tags, much worse than what one sees with CCM [203] and other modes of operation. The defect is not an artifact of the analysis, as Ferguson's attack makes clear.

1.9.8 I call attention to the problem of properly defining authenticated encryption for the

setting in which authentication tags may be short, acknowledging the deficiency of existing definitions for this setting, where one should not "give up" (regard the adversary as having won) when the adversary forges, for example, a one-byte tag.

Some of these matters are appropriate targets for follow-on academic papers.

Chapter 2

Preliminaries

2.1. Notation. We aim to use notation standard enough that it will not be much noticed or questioned by likely readers of this report. By "strings" we always mean (finite) binary strings. The empty string is written ε. The bit-length of a string X is written $|X|$. The byte-length of a byte string X is written $|X|_8$. Concatenation of strings A and B is written as either $A \parallel B$ or $A\,B$. We sometimes use conventional formal-language notation, like $\{0,1\}^*$ for binary strings or $(\{0,1\}^n)^+$ for strings having a positive multiple of n bits. We occasionally use hexadecimal notation, as in `0x36` for the byte 00110110. The first t bits of a string X of length at least t is written $\mathrm{MSB}_t(X)$, while the last t bits are $\mathrm{LSB}_t(X)$. We write $A \oplus B$ for the bitwise exclusive-or of two equal-length strings. We also use this notation for the xor of unequal-length strings, where the understanding is to drop rightmost bits of the longer string until the two strings are equal, and then bitwise xor.

We write $a \xleftarrow{\$} \mathcal{X}$ for the experiment of randomly sampling from a space $\mathcal{X}$ and assigning the result to a. Either $\mathcal{X}$ will have a distribution associated to it or $\mathcal{X}$ will be finite and we mean the uniform distribution.

Adversaries will typically be understood as (possibly probabilistic) algorithms with access to oracles. We write oracles that an adversary $\mathcal{A}$ may access as in $\mathcal{A}^F$. When we write an expression like $\Pr[\mathcal{A}^{F_K} \Rightarrow 1]$ or $\Pr[\mathcal{A}^{F_K(\cdot)} \Rightarrow 1]$ or $\Pr[K \xleftarrow{\$} \mathcal{K} :\ \mathcal{A}^{F_K} \Rightarrow 1]$ we mean the probability that $\mathcal{A}$ outputs 1 when it is run with an oracle F_K. What the probability is over will either be explicit in the bracketed experiment or implicit in the reader's understanding of the notation.

If $i \in [0..2^{n-1}]$ is a number we write $[\, i \,]_n$ for the n-bit string that encodes i as a binary number.

We walk a fine line between trying to preserve the variable names and other notation found in the defining standards and trying to use either more uniform or more elegant notation throughout this report. Be sensitive to the possibility that notation, as well as descriptive approach, may have been silently changed.

2.2. Blockciphers. Modes of operation are usually modes of operation of a *blockcipher*, so we now review the usual definitions for blockciphers and their security. Refer to [17, 18, 126, 128] for early treatments. A blockcipher is a function $E\colon \mathcal{K}\times\{0,1\}^n \to \{0,1\}^n$ such that $E_K(\cdot) = E(K,\cdot)$ is a permutation on $\{0,1\}^n$. The inverse to the blockcipher E is $D = E^{-1}$ defined by $D_K(Y)$ being the unique $X \in \{0,1\}^n$ such that $E_K(X) = Y$. A typical blockcipher is AES. Actually, to match the definition just given, the AES spec [158] would be understood as any defining *three* blockciphers, AES-128, AES-192, and AES-256, one for each of three permitted key length. Typically a scheme's *parameters*, like the key length k for AES, must be fixed for the scheme,

as described in the spec, to match the syntax we employ in a definition.

The usual way to quantify the security of a blockcipher $E\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ works as follows. Choose a random $K \xleftarrow{\$} \mathcal{K}$ and a random permutation π on n-bits. An adversary $\mathcal{A}$ is given blackbox access either to E_K or to π. The adversary tries to guess which kind of object it has. We let

$$\mathbf{Adv}^{\mathrm{prp}}_E(\mathcal{A}) \;=\; \Pr[K \xleftarrow{\$} \mathcal{K}\colon\ \mathcal{A}^{E_K(\cdot)} \Rightarrow 1] - \Pr[\pi \xleftarrow{\$} \mathrm{Perm}(n)\colon\ \mathcal{A}^{\pi(\cdot)} \Rightarrow 1] \tag{2.1}$$

where $\mathrm{Perm}(n)$ denotes all the permutation on n-bit strings.

An alternative measure of security for a blockcipher compares it against a random function instead of a random permutation. In that case we set

$$\mathbf{Adv}^{\mathrm{prf}}_E(\mathcal{A}) \;=\; \Pr[K \xleftarrow{\$} \mathcal{K}\colon\ \mathcal{A}^{E_K(\cdot)} \Rightarrow 1] - \Pr[\rho \xleftarrow{\$} \mathrm{Func}(n,n)\colon\ \mathcal{A}^{\rho(\cdot)} \Rightarrow 1] \tag{2.2}$$

where $\mathrm{Func}(n,n)$ denotes the set of all functions from n-bit strings to n-bit strings. It is a standard result that $\Pr[\mathcal{A}^{\pi} \Rightarrow 1] - \Pr[\mathcal{A}^{\rho} \Rightarrow 1] \le q^2/2^{n+1}$ for any adversary $\mathcal{A}$ that asks at most q queries. This makes the PRP and PRF notions of advantage close: $|\mathbf{Adv}^{\mathrm{prp}}_E(\mathcal{A}) - \mathbf{Adv}^{\mathrm{prf}}_E(\mathcal{A})| \le q^2/2^{n+1}$ if $\mathcal{A}$ asks q or fewer queries. The observation is sometimes known as the PRP/PRF Switching Lemma [27].

Yet another important security notion for blockciphers strengthens the requirement for resembling a random permutation by giving the adversary oracles not only for the forward direction of the cipher or the random permutation, but for the backwards direction, too. This is sometimes called the "strong" notion of PRP security. It amounts to permitting a chosen-ciphertext attack (in addition to a chosen-plaintext attack). The definition of advantage becomes

$$\mathbf{Adv}^{\pm\mathrm{prp}}_E(\mathcal{A}) \;=\; \Pr[K \xleftarrow{\$} \mathcal{K}\colon\ \mathcal{A}^{E_K(\cdot),\, E_K^{-1}(\cdot)} \Rightarrow 1] - \Pr[\pi \xleftarrow{\$} \mathrm{Perm}(n)\colon\ \mathcal{A}^{\pi(\cdot),\, \pi^{-1}(\cdot)} \Rightarrow 1]. \tag{2.3}$$

It is easy to see that good $\pm$prp-security implies good prp-security, but that the reverse does not hold.

It is strongly believed that blockciphers like $E = \mathrm{AES}$ are good in the PRP and strong-PRP senses of the word: "reasonable" adversaries have only a "small" value $\mathbf{Adv}^{\mathrm{prp}}_E(\mathcal{A})$ or $\mathbf{Adv}^{\pm\mathrm{prp}}_E(\mathcal{A})$. In the concrete-security tradition, the approach that makes makes most sense when dealing with blockciphers, we usually leave it at that, not trying to define *reasonable* or *small*. If one did want to specify an explicit assumption, one would say something like "we think that $\mathbf{Adv}^{\pm\mathrm{prp}}_{\mathrm{AES128}}(\mathcal{A}) < 2^{-10}$ if A asks at most 2^{50} queries and uses at most 2^{90} time" (with time understood to include the adversary's description size and some particular computational model being fixed). But we certainly do not know how to prove any such statement, and making concrete conjectures like the one just given does not seem to have any real value.

There are other notions of blockcipher security, but the PRP, PRF, and strong-PRP notions have proven to be the most productive. Weaker notions of blockcipher security, like unpredictability of the blockcipher, seem to be inadequate for supporting efficient schemes with strong security proofs. In particular, none of the modes looked at within this report would be secure if one only demanded the blockcipher to be unpredictable. Similarly, there are stronger notions for blockcipher-security, like security with respect to related-key attacks. But such assumptions seem to be overkill for using the blockcipher to achieve standard privacy and authenticity aims. Insisting that blockcipher-based constructions make use of nothing beyond PRP or strong-PRP security is a way to impose some restraint in design, and a way to make sure that one is not assuming unrealistically much.

Blockciphers of the sort the reader probably has had in mind on reading this section are what one might call *conventional* blockciphers; in particular: blocksize n is some fixed and

rather small number, like $n = 64$ or $n = 128$, and the blockcipher is a "true" primitive, not something built up from something else. We warn the reader that, in this report, we are going to be rather more catholic in what we are willing to call a blockcipher. In particular, ECB mode, enciphering an arbitrary string in $(\{0,1\}^n)^+$, is *also* a blockcipher, at least if you forgive the transgression that its domain contains strings of multiple different lengths (even then, the old definition would apply for each possible blocksize n). Now as a blockcipher, ECB may disappoint, since it does not achieve good security in the prp-sense we have defined. But it is still a blockcipher.

A bit more carefully, to regard a mode like ECB as a blockcipher one treats the syntax of a blockcipher just a bit more liberally, now defining a blockcipher as a function $\mathcal{E}\colon\ \mathcal{K} \times \mathcal{X} \to \mathcal{X}$ where $\mathcal{X} \subseteq \{0,1\}^*$ is the message space and $\mathcal{E}_K(\cdot) = \mathcal{E}(K, \cdot)$ is a length-preserving permutation. The inverse to the blockcipher $\mathcal{E}$ is $\mathcal{D} = \mathcal{E}^{-1}$ defined by $\mathcal{D}_K(Y)$ being the unique $X \in \{0,1\}^n$ such that $\mathcal{E}_K(X) = Y$. To define the prp and $\pm$prp security notions for blockcipher $\mathcal{E}\colon\ \mathcal{K} \times \mathcal{X} \to \mathcal{X}$ one adjusts (2.1) and (2.3) to

$$\mathbf{Adv}^{\mathrm{prp}}_{\mathcal{E}}(\mathcal{A}) \quad = \quad \Pr[K \xleftarrow{\$} \mathcal{K}\colon\ \mathcal{A}^{\mathcal{E}_K(\cdot)} \Rightarrow 1] - \Pr[\pi \xleftarrow{\$} \mathrm{Perm}(\mathcal{X})\colon\ \mathcal{A}^{\pi(\cdot)} \Rightarrow 1] \tag{2.4}$$

and

$$\mathbf{Adv}^{\pm\mathrm{prp}}_{\mathcal{E}}(\mathcal{A}) = \Pr[K \xleftarrow{\$} \mathcal{K}\colon\ \mathcal{A}^{\mathcal{E}_K(\cdot),\, \mathcal{E}_K^{-1}(\cdot)} \Rightarrow 1] - \Pr[\pi \xleftarrow{\$} \mathrm{Perm}(\mathcal{X})\colon\ \mathcal{A}^{\pi(\cdot),\, \pi^{-1}(\cdot)} \Rightarrow 1] \tag{2.5}$$

where $\mathrm{Perm}(\mathcal{X})$ denotes the set of all length-preserving permutations on $\mathcal{X}$. We comment that we changed the blockcipher "E" to an "$\mathcal{E}$" as we will attempt to use the former symbol to suggest a conventional blockcipher and the latter for a higher-level encryption scheme (including a blockcipher like ECB, built out of an underlying blockcipher).

2.3. Modes of operation. When a blockcipher is used in some scheme to accomplish a higher-level goal, we call the blockcipher-using scheme a (blockcipher-based) *mode of operation*. Usually one sees the term *mode of operation* without the "blockcipher" qualification; it is *understood* that we are basing the mode on a blockcipher. Besides using the blockcipher, the mode may use other simple tools, like simple bit manipulations, xor operations, message padding, and even some finite-field arithmetic. If one started to use quite complex operations—something like a modular exponentiation or an elliptic-curve computation—one would probably draw the line and say that this no longer looked like a mode of operation: there is no precise meaning ascribed to the term *modes of operation* of a blockcipher, but we expect, somehow, that use of the blockcipher is the dominant thing that is going on.

The goal of different modes of operation varies. We will be looking at a variety of goals in this report, but all of them involving privacy and authenticity.

Besides the blockcipher modes of operation, we will also be looking at HMAC, which is normally understood as being a mode of operation of a *hash function* rather than a blockcipher mode. The idea is the same, however; the higher-level goal (here, making a message authentication code / an arbitrary-input-length PRF) is accomplished from the lower-level primitive (the cryptographic hash function).

2.4. Reduction-based approach. We will mostly be following the reduction-based approach of modern cryptography, the branch of cryptography most strongly associated to Goldwasser and Micali's seminal work [74]. We follow the "practice-oriented" variant of this approach, most strongly associated to Bellare and Rogaway [14, 17, 18, 25].

The idea, in this setting, is as follows. We have some blockcipher mode of operation, Π, that we would like to prove secure. The scheme Π depends on a blockcipher E, so you have to

specify E before $\Pi[E]$ becomes a concrete scheme. The mode is supposed to accomplish some goal—say, for concreteness, the goal might be message authentication. So we formally *define* the goal—for concreteness, the "MAC" formulation of security for message authentication. Formalizing the MAC goal means that we associate to any adversary $\mathcal{A}$ and any message authentication code F a real number $\mathbf{Adv}^{\mathrm{mac}}_{F}(\mathcal{A}) \in [0,1]$ that measures how good a job the adversary $\mathcal{A}$ does in "breaking" the MAC. An advantage of 0 means "terrible job" and an advantage of 1 means "great job." Now to evidence that we have used E well in making our mode $\Pi[E]$ we prove a theorem to capture the intent that

> *If E is a good blockcipher, then $\Pi[E]$ is a good MAC.*

Taking the contrapositive, we want a theorem that says that

> *If $\Pi[E]$ is a poor MAC, then E is a poor blockcipher.*

Making the role of the adversary more explicit:

> *If there is a reasonable adversary $\mathcal{A}$ that does well at breaking $\Pi[E]$ as a MAC, then there is a reasonable adversary B that does well at breaking E as a blockcipher.*

Inserting our named notions of security and explicitly emphasizing the constructive nature of our enterprise:

> *Given an adversary $\mathcal{A}$ that achieves advantage $\mathbf{Adv}^{\mathrm{mac}}_{\Pi[E]}(\mathcal{A})$ there is a corresponding adversary $\mathcal{B}$, explicitly determined from the description of $\mathcal{A}$, where adversary $\mathcal{B}$ is about as efficient as $\mathcal{A}$ and where $\mathcal{B}$ gets advantage $\mathbf{Adv}^{\mathrm{prp}}_{E}(\mathcal{B})$ in attacking E that is about as large as $\mathbf{Adv}^{\mathrm{mac}}_{\Pi[E]}(\mathcal{A})$.*

A careful theorem statement gives an explicit formula that lower bounds $\mathbf{Adv}^{\mathrm{prp}}_{E}(\mathcal{B})$ in terms of $\mathbf{Adv}^{\mathrm{mac}}_{\Pi[E]}(\mathcal{A})$. The theorem also gives formulas to explain how the computational resources of $\mathcal{B}$ compare to those for $\mathcal{A}$. We hope for "tight" reductions—meaning, in this case, that $\mathbf{Adv}^{\mathrm{prp}}_{E}(\mathcal{B})$ should be close to $\mathbf{Adv}^{\mathrm{mac}}_{\Pi[E]}(\mathcal{A})$ and $\mathcal{B}$ should be almost as efficient as was $\mathcal{A}$. The formulas make manifest *how* tight the reduction be.

In the approach we suggest and implicitly use throughout this report, there are is no asymptotics. No polynomial times, expected polynomial time, negligible functions, or the like. Anything we want to say about how one scheme's security is related to another can be stated precisely, and in a way that makes sense for functions with finite domains.

The point of giving a reduction is to bootstrap assurance: our belief in the security of a blockcipher is leveraged into a belief about the blockcipher mode. The point in having explicit and quantitatively precise reductions is that we can answer just *how* secure the mode has been proven to be relative to the blockcipher's assumed strength.

Because we aim to base designs for modes of operation on the PRP or strong-PRP assumption, things like related-key attacks on AES [33, 34] tend to be *irrelevant* when looking at the security of a proven-secure AES-based mode. The alleged weakness of the blockcipher is not relevant to our confidence that the mode does its job; all that matters is that AES be good as a PRP. At least with respect to this use of AES, we ought not to care if AES is subject to an effective related-key attack.

It should be emphasized that a proof that a mode is secure if its underlying blockcipher is a PRP provides *no* assurance of security if the underlying blockcipher is *not* a PRP. Also, the

meaning of the mode being "secure" is always something quite specific: the mode does well with respect to some specified definition D. Absent other evidence, there will be *no* reason to believe that the scheme does well with respect to some definition D' that differs—even a little—from definition D. This is one reason cryptographers study security relations among definitions; we want to make sure that our definitions are reasonably robust.

2.5. Information-theoretic arguments. At the core of almost every reduction-based proof for a blockcipher mode of operation is a proof that has nothing to do with complexity-theoretic cryptography. One regards E_K, for a random key K, as a uniform random permutation π. One then proves, for this idealized blockcipher, that the mode of operation *definitely* does well. If the mode is for a MAC, say, then you get a bound on $\mathbf{Adv}^{\mathrm{mac}}_{\Pi[\mathrm{Perm}(n)]}(\mathcal{A})$ for any adversary that asks a certain number of queries, those queries of specified lengths. One makes *no* assumptions about how long adversary $\mathcal{A}$ may run. Working in the information-theoretic realm makes things vastly less provisional: we *know* how well our adversaries can do, and we can give concrete results to show it.

After giving the information-theoretic "core" of a reduction, we must generally go back and describe what, concretely, it means when the blockcipher E is *not* ideal. Inevitably—well, at least for all of the examples that arise in this report—this step is quite routine, and only ends up, in a theorem statement, taking on a term like $\mathbf{Adv}^{\mathrm{prp}}_{E}(\mathcal{B})$. When we describe the "quality" of a reduction, we often ignore this final step, because it never (again, in the context of the modes of this report) has any bearing on how good a job one has done in demonstrating a mode secure. When describing the quality of a reduction within the information-theoretic setting, this is done not because the final result is in an idealized setting, it is only for simplicity of exposition, and to focus on what matters most.

2.6. Other approaches to looking at the security of a mode. In the *ideal-cipher model*—see §II.6—we don't bother with reductions at all. Instead, we treat the underlying blockcipher $E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ as a family of random permutations on n bits, each one named by a k-bit key. The adversary is given black-box access to $E(K,X)$ and its inverse $D(K,Y)$. Because modes of operation most often *can* be analyzed in the "standard," reduction-based approach, it is the one that cryptographers generally prefer. The bias is ultimately rooted in a classical belief in the danger of abstracting away computational aspects of algorithm.

Many cryptographers have let the entire provable-security tradition pass them by and continue to use the more classical, attack-based approach. Throughout this report, we do sometimes describe attacks on the considered modes of operation. We have mixed feelings on attack-based study of modes. Unless one steps outside of the model (eg, in looking at timing attacks, or looking at attacks that break the "rules" of the mode, as when one requires repetition of a nonce), one *knows*, once a scheme has been proven secure, that the attack will not be more effective than that which is guaranteed by the bound. If the bound is good and the model is not being subverted, an attack-based analysis of a mode is, then, an exercise in futility. But, opposing this view, one cannot say that a mode of operation is fully understood until one has both security upper *and* lower bounds, and found that they essentially match. Attacks therefore tell you how much room there is, or isn't, in improving ones bounds. Additionally, looking at modes—or any cryptographic scheme—from an alternative perspective can inspire new insights and questions. So we are certainly not "opposed" to trying to break modes, even proven-secure ones, by looking at attacks. One should simply understand the limits of what might be shown in the presence of sound and strong proofs. And ultimately it is proofs—not the inexistence of effective attacks—that should be the thing that engenders confidence in a mode's worth.

Part I

Confidentiality Modes

Part I

Confidentiality Modes

I.1. Overview. The six modes of operation reviewed in Part I of this document are all focused on achieving message confidentiality. Five of the six modes—CBC, CFB, OFB, CTR, and XTS—can be understood as having "syntax" that looks like this:

$$C = \mathcal{E}_K^{IV}(P) \text{ where } IV \text{ is either a } \textbf{nonce}, \text{ a } \textbf{random value}, \text{ or a } \textbf{tweak}. \tag{2.6}$$

For one of the six confidentiality modes—ECB—there is no initialization vector (IV). The absence of an IV can, for consistency, be understood as the silent provisioning of a fixed IV, say the empty string. But we will not take this view, regarding ECB as a fundamentally different "kind" of object from the other five modes. We call a mechanism $\mathcal{E}$ with a syntax as in $\mathcal{E}_K^{IV}(P)$ an *IV-based encryption scheme*. A more precise definition will be given later.

The set of possibilities from which IV is drawn in (2.6), the *IV space*, differs across the different modes, as do the sets from which one draws the key K (the *key space*) and the plaintext P (the *message space*). Each of the six confidentiality modes implicitly depends on an underlying blockcipher

$$E\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n\,. \tag{2.7}$$

The permitted blockciphers, the *blockcipher space*, one might say, also varies across the different modes. Altogether, the syntax for our six confidentiality schemes can be summarized as follows:

Mode	**key space**	**message space**	**IV space**	**blockcipher space**
ECB	$\mathcal{K}$	$(\{0,1\}^n)^+$	N/A	AES, Skipjack, or TDEA
CBC	$\mathcal{K}$	$(\{0,1\}^n)^+$	$\{0,1\}^n$	AES, Skipjack, or TDEA
CFB	$\mathcal{K}$	$(\{0,1\}^s)^+$	$\{0,1\}^n$	AES, Skipjack, or TDEA
OFB	$\mathcal{K}$	$\{0,1\}^*$	$\{0,1\}^n$	AES, Skipjack, or TDEA
CTR	$\mathcal{K}$	$\{0,1\}^*$	$(\{0,1\}^n)^+$	AES, Skipjack, or TDEA
XTS	$\mathcal{K} \times \mathcal{K}$	$\{0,1\}^{\geq n}$	$\{0,1\}^n$	AES (only AES-128 or AES-256)

For CFB mode, the number s ($1 \leq s \leq n$) one sees showing up in the message space is a parameter of the scheme. For CTR mode, the number of components in the provided IV must be equal to the block length of the provided plaintext, $m = \lceil |P|/n \rceil$.

I.2. Possible goals. When using the term "IV-based encryption scheme" I am not implying any particular security property. Correspondingly, there is no such thing as being "good" in the

sense of IV-based encryption. Here, quite broadly, are three security goals to which an IV-based encryption scheme might aspire:

Probabilistic encryption. An IV-based encryption can be secure as a *probabilistic* encryption scheme. There are multiple ways to formalize this, but in all of them the IV is assumed to be selected uniformly at random, chosen anew with each message that is encrypted. We may here denote the IV by the letter R, writing $C = \mathcal{E}_K^R(P)$. The IV is, in this setting, not under the user's (nor the adversary's) control. The random IV is normally assumed to accompany the ciphertext. Indeed in many treatments in the cryptographic literature, the random IV would be considered as *part* of the ciphertext (but we will not formalize things in this way). Probabilistic symmetric encryption schemes were first formalized and investigated by Bellare, Desai, Jokipii, and Rogaway [14], building on Goldwasser and Micali's approach in the public-key setting [74].

Nonce-based encryption. An IV-based encryption scheme might be secure as a *nonce-based* encryption scheme. There are, again, multiple way to formalize this, but in all of them the IV *is* under the user's control. What is expected of him is that he provides a *new* IV—a *nonce*—with every message he encrypts. A *counter* (one that is forbidden to "wrap around") is the prototypical nonce. A random value can also be used as a nonce, assuming the random value is long enough so that collisions will almost never arise. In the nonce-based setting we sometimes denote the IV by the letter N, writing $C = \mathcal{E}_K^N(P)$. Nonce-based symmetric encryption was first formalized by Rogaway [180], although the idea was no doubt long present in the "folklore" of applied cryptography.

Tweakable blockcipher. An IV-based encryption scheme might be secure as a *tweakable blockcipher*. Now the IV is called a "tweak," (or, less often, a "salt" or "diversity parameter"). Correspondingly, we write $C = \mathcal{E}_K^T(P)$, giving the IV yet another letter. One encrypts a plaintext P using key K and tweak T to get a ciphertext $C = \mathcal{E}_K^T(P)$. You can also reverse the map, letting $P = \mathcal{D}_K^T(C)$. The length of plaintext P and the ciphertext C must again be equal. Usually what we want is, at least, that, for each tweak T, the permutation $\mathcal{E}_K(\cdot)$ looks like a random permutation to one who does not know the key K. Tweakable blockciphers were first formalized by Liskov, Rivest, and Wagner [124].

I.3. SemCPA notions. When we speak of **SemCPA security** ("semantic security with respect to a chosen-plaintext attack") we are assuming the setting of a probabilistic or nonce-based encryption scheme and a formalization of security that captures an adversary's inability to break the scheme when launching a chosen-plaintext attack (CPA). With the SemCPA label we do not mean any *specific* security definition; we mean any of several security definitions along these lines.

The particular SemCPA security definitions we focus most on is what we call **ind$**-security (for the probabilistic setting) and **IND$**-security (for the nonce-based one). If we say that a scheme is SemCPA secure, we mean that it is probabilistic or nonce-based and that it does well with respect to achieving at least one of the formulations for semantic security with respect to a SemCPA attack. For example, the scheme might do well with respect to the ind$ security definition.

When we speak of a scheme aiming to achieve (only) privacy or confidentiality, or as being a privacy-only or confidentiality-only scheme, what we mean is that it targets some SemCPA security definition and not something stronger.

One could also speak of SemCCA notions for security. Such formulations invariably provide the adversary a decryption oracle, in addition to whatever other oracles or capabilities the adversary may have.

I.4. Quick preview. Very briefly, we will be seeing that CBC, CFB, and OFB are all SemCPA secure as probabilistic encryption schemes. None of the methods are SemCCA secure as probabilistic encryption schemes. We will see that none of them are SemCPA secure as nonce-based schemes, either. The NIST spec suggests that a nonce can be used for the IV of OFB [61, p. 13 and p. 20, paragraph 5], but this is not enough with respect to our definitions. The NIST spec explains that CBC and CFB are not secure if the IV is merely a nonce and it spells out a modification to those modes [61, Appendix C] that seems aimed at making them nonce-based. We name these CBC and CFB variants ♯CBC and ♯CFB. We show that ♯CBC and ♯CFB are SemCPA insecure as nonce-based encryption schemes. We will be explaining all of these claims in Chapter 4.

CTR mode is the only mode we review that is SemCPA secure as a nonce-based encryption scheme. But to fit CTR into this nonce-based mold we have to be pretty generous in how we formalize our nonce-based notion: the nonce needs to be a vector of strings, one for each block, and reuse of a nonce is understood to include reuse of any of the nonce's blocks.

None of the confidentiality modes reviewed in this report are good in the sense one would want of a tweakable blockcipher. Just the same, understanding tweakable blockciphers will be necessary to understand our discussion of XTS.

It should be emphasized that there are security notions for an IV-based encryption scheme that go beyond SemCPA security. We mentioned SemCCA security. Other goals include non-malleability (NM) and an inability to create ciphertexts that decrypt to valid plaintexts (authenticated encryption).

I.5. Syntax of an IV-based encryption scheme. We now formalize what an IV-based encryption scheme actually *is*, as a syntactic object.

It has become traditional to regard blockciphers as fixed functions (or algorithms), but to define more "complicated" kinds of encryption schemes as tuples of algorithms, as in $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. To simplify and unify matters a bit we will instead formalize IV-based encryption schemes more like blockciphers: as functions, not as tuples.

Formally, we will say that an IV-based encryption scheme is a function

$$\mathcal{E} \colon \mathcal{K} \times \mathcal{IV} \times \mathcal{X} \to \mathcal{X} \; . \tag{2.8}$$

We will usually write $\mathcal{E}_K^{IV}(P)$ instead of $\mathcal{E}(K, IV, P)$. The sets $\mathcal{K}$, $\mathcal{IV}$, and $\mathcal{X}$ are called the *key space*, the *IV space*, and the *message space*. The message space $\mathcal{X} \subseteq \{0,1\}^*$ must be a set of strings. The key space $\mathcal{K}$ is usually a finite set, in which case it is understood to have associated to it the uniform distribution. It would be fine for $\mathcal{K}$ to be infinite as long as there is an understood distribution. Each $IV \in \mathcal{IV}$ is assumed to be a sequence of strings, $\mathcal{IV} \subseteq (\{0,1\}^*)^*$. This generality is only needed for CTR mode; in all other case, the IV is a *single* string, $IV \in \{0,1\}^*$. We demand that, for any $K \in \mathcal{K}$ and $IV \in \mathcal{IV}$, the function $\mathcal{E}_K^{IV}(\cdot)$ is a length-preserving permutation on $\mathcal{X}$. As such, there is a unique function $\mathcal{D}$ corresponding to $\mathcal{E}$, the *decryption* function $\mathcal{D} = \mathcal{E}^{-1}$ for the encryption scheme $\mathcal{E}$, defined by $\mathcal{D}_K^{IV}(C) = P$ exactly when $\mathcal{E}_K^{IV}(P) = C$.

It would not be a problem to weaken the "length-preserving permutation" requirement to allow some arbitrary injective function—and such a relaxation would in fact be necessary if, for

example, mandatory 10^* padding was assumed prior to CBC encryption, as described in [61, Appendix A].

When time-complexity arises with respect to an IV-based encryption scheme one should switch to regarding it as an algorithm, not just a function. Of course any *practical* IV-based encryption scheme $\mathcal{E}$ will need to have an efficient algorithmic realization, and there should be an efficient algorithmic realization for the decryption direction $\mathcal{D} = \mathcal{E}^{-1}$ of $\mathcal{E}$, too.

We note that for the receiver to recover the plaintext P from the ciphertext C under the key K, she must present the same value IV as before; although we do not consider the IV to be part of the ciphertext, it must be known to the party that wants to decrypt. So, in applications, either it will need to be presented to the party that will decrypt alongside the ciphertext, or else it must be recoverable by her by some other means.

As we have indicated, there are multiple security notions that can be overlaid on the syntax of an IV-based encryption scheme. We will define two, and sketch more.

I.6. Security of a probabilistic encryption scheme. First we consider SemCPA security notions for IV-based encryption schemes where the IV is taken to be a uniformly *random* string. Here we assume, for simplicity, that $\mathcal{IV} = \{0,1\}^n$ consists of strings of some one fixed length. While the syntax of an IV-based encryption scheme $\mathcal{E}$ shows three strings being provided to the mechanism, in the current context the "user" is expected to provide only two of them, the key K and the plaintext P. The mechanism internally selects the third string, sampling $R \xleftarrow{\$} \{0,1\}^n$ before it computes $C \leftarrow \mathcal{E}_K^R(P)$ and returns $\mathcal{C} = R \parallel C$. We give a very strong notion of security for this setting—the notion of *indistinguishability from random bits*.

Fix an IV-based encryption scheme $\mathcal{E} \colon \mathcal{K} \times \{0,1\}^n \times \mathcal{X} \to \mathcal{X}$ and let $\mathcal{A}$ be an adversary. We consider two different oracles with which $\mathcal{A}$ may communicate:

Real. First, a random key $K \xleftarrow{\$} \mathcal{K}$ is selected. After that, the oracle behaves as follows. On input of a plaintext $P \in \mathcal{X}$, the oracle selects a random value $R \xleftarrow{\$} \{0,1\}^n$, defines $C \leftarrow \mathcal{E}_K^R(P)$, and returns to the adversary $\mathcal{C} = R \parallel C$. If $P \notin \mathcal{X}$ the oracle responds with Invalid. We denote this oracle $\mathcal{E}_K(\cdot)$.

Random. On input of a plaintext $P \in \mathcal{X}$ the oracle selects random strings $R \xleftarrow{\$} \{0,1\}^n$ and $C \xleftarrow{\$} \{0,1\}^{|P|}$, and returns $\mathcal{C} = R \parallel C$. If $P \notin \mathcal{X}$ the oracle responds with Invalid. We denote this oracle $\$(\cdot)$.

We define

$$\mathbf{Adv}^{\mathrm{ind\$}}_{\mathcal{E}}(\mathcal{A}) \quad = \quad \Pr[\mathcal{A}^{\mathcal{E}_K(\cdot)} \Rightarrow 1] - \Pr[\mathcal{A}^{\$(\cdot)} \Rightarrow 1] \tag{2.9}$$

as the difference in probabilities for $\mathcal{A}$ outputting 1 in the two settings. Informally, an IV-based encryption scheme is secure in the ind\$-sense if $\mathbf{Adv}^{\mathrm{ind\$}}_{\mathcal{E}}(\mathcal{A})$ is "small" whenever $\mathcal{A}$ is "reasonable." We will have no need to precisely define *small* or *reasonable*; concrete security reductions give more precise assertions.

There are multiple weakenings of the ind\$-security that have been considered in the literature. Let us sketch a few of these other SemCPA notions, as drawn from Bellare, Desai, Jokipii, and Rogaway [14] and based, in part, on Goldwasser and Micali [74].

1. With the **indistinguishability** notion of security the adversary doesn't have to distinguish ciphertexts from random bits—she has, instead, to distinguish ciphertexts from the

encryptions of random bits. Here one assumes that the message space $\mathcal{X}$ has the structure that if $P \in \mathcal{X}$ then so is every other string of $|P|$ bits. In a slight variation of this notion, what is given to the adversary is not the encryption of random bits (when not the encryption of the actual query), but the encryption of 0-bits, the number of 0-bits being the length of the adversary's query. Here one assumes that the message space $\mathcal{X}$ has the structure that, if $P \in \mathcal{X}$, then so is $0^{|P|}$.

2. With the **left-or-right** notion of security, the oracle begins by randomly selecting a key $K \xleftarrow{\$} \mathcal{K}$, as before, and it also selects a bit $b \xleftarrow{\$} \{0,1\}$. Adversary $\mathcal{A}$ now asks a sequence of query pairs, each such pair of the form (P_0, P_1) where strings P_0 and P_1 are equal-length messages in $\mathcal{X}$. In response to any such query, the oracle computes $\mathcal{C} \xleftarrow{\$} \mathcal{E}_K(P_b)$, as defined before, returning this string. When satisfied, the adversary outputs a bit b'. The adversary's advantage is defined the probability that $b' = b$, minus 1/2 to account for random guessing. Usually one further multiplies by two and subtracts one, just to normalize advantage values to the range $[-1, 1]$ (or $[0, 1]$, if one takes the absolute value or ignores negative advantages).

3. With the **find-then-guess** notion of security, the adversary $\mathcal{A}$ runs in two stages, Find then Guess. During both it may access an oracle for $\mathcal{E}_K(\cdot)$ that works as above. The result of the Find stage is a pair of equal-length strings $P_0, P_1 \in \mathcal{X}$ and some arbitrary saved state s (capturing what the adversary wants to "remember"—we are regarding the Find and Guess stages of $\mathcal{A}$ as two distinct algorithms). The experiment then chooses a random bit $b \xleftarrow{\$} \{0,1\}$ and sets $C \xleftarrow{\$} \mathcal{E}_K(P_b)$. During the adversary's Guess stage, it is provided as input (C, s). We look at the probability that the adversary ends its Guess stage by outputting the bit b' that is the same as b. The adversary's advantage is this probability minus 1/2 (or else this value re-normalized to $[0, 1]$ or $[-1, 1]$, just as before).

4. Under the **semantic security** notion, the adversary $\mathcal{A}$ runs in two stages, Select then Predict. During both it may access an oracle $\mathcal{E}_K(\cdot)$ as above. The result of the Select stage is a tuple $(\mathcal{M}, s)$ where s is a string and $\mathcal{M}$ encodes a probabilistic algorithm that returns a sample within the domain $\mathcal{X}$ of the encryption scheme, all samples that may be output having the same fixed length. After this tuple is output, the experiment selects $b \xleftarrow{\$} \{0,1\}$, samples $P_0, P_1 \xleftarrow{\$} \mathcal{M}$, computes $C \xleftarrow{\$} \mathcal{E}_K(P_b)$, and returns to the adversary, now running in its Predict stage, the pair (C, s). The adversary wants to figure out *something* about the plaintext P_b that is the decryption of ciphertext C. To that end, it terminates with an output of (f, y) where f encodes a string-valued function on $\mathcal{X}$ and y is a string. The adversary's advantage is the probability that $f(P_b) = y$ minus the probability that $f(P_{1-b}) = y$. The definition just sketched is a little stronger than that in [14], insofar as we have postponed the output of f.

The important thing to understand is not so much the details or intuition for each alternative definition, but the fact that all of these alternative notions are implied by ind$-security—and by tight reductions. As a consequence, if we want to show a strong SemCPA property for a probabilistic (IV-based) encryption, it is enough to focus on just the ind$ notion—directly establishing any of the other notions would then be pointless. In fact, the indistinguishability-from-random-bits notion is properly *stronger* than all four of the alternative notions sketched above.

NIST document SP 800-38A often refers to the IV as being unpredictable rather than random. It is possible to weaken our notion of probabilistic encryption in exactly that direction, taking the *IV* to be generated by a *source* that, after q samples, the next one can be correctly

predicted with probability at most $\varepsilon(q,t)$ by any algorithm running in time (plus description size) at most t. We do not pursue this added complexity, which would seem to add little insight on any of the modes. We concur, without trying to formally show theorems, that all of the SP 800-38A modes that are secure as probabilistic encryption schemes—namely, CBC, CFB, and OFB—will remain secure if the *IV*is not perfectly random, but only unguessable. A quantitative statement of such results would "give up" in the ind$ advantage an amount proportional to the $\varepsilon(q,t)$ value defined above.

I.7. Security of a nonce-based encryption scheme. We emphasize that, under our treatment of IV-based encryption schemes, both probabilistic and nonce-based security notions apply: these are alternative security notions for the *same* kind of object. What is different is what we assume is *done* with the IV. In the ind$-security notion, we formalized that the IV is random and not under the user's control. In the nonce-based setting the opposite is assumed: the IV *is* under the user's control, but he must execute a modicum of restraint in its provisioning, never providing the same IV twice. The formalization follows.

Once again fix an IV-based encryption scheme $\mathcal{E}\colon \mathcal{K}\times\mathcal{IV}\times\mathcal{X}\to\mathcal{X}$ and let $\mathcal{A}$ be an adversary. We consider two different oracles with which $\mathcal{A}$ may communicate:

REAL. First, a random key $K \xleftarrow{\$} \mathcal{K}$ is selected. After that, the oracle behaves as follows. On input (N,P), an IV $N\in\mathcal{IV}$ and a plaintext $P\in\mathcal{X}$, the oracle computes and returns $C\leftarrow\mathcal{E}_K^N(P)$. If $N\notin\mathcal{IV}$ or $P\notin\mathcal{X}$ the oracle responds with Invalid. We denote this oracle $\mathcal{E}_K(\cdot,\cdot)$.

RANDOM. On input (N,P), an IV $N\in\{0,1\}^n$ and a plaintext $P\in\mathcal{X}$, the oracle computes $C\xleftarrow{\$}\{0,1\}^{|P|}$ and returns C. If $N\notin\{0,1\}^n$ or $P\notin\mathcal{X}$ the oracle responds with Invalid. We denote this oracle $\$(\cdot,\cdot)$.

We define

$$\mathbf{Adv}^{\mathrm{IND\$}}_{\mathcal{E}}(\mathcal{A}) \;=\; \Pr[\mathcal{A}^{\mathcal{E}_K(\cdot,\cdot)}\Rightarrow 1]-\Pr[\mathcal{A}^{\$(\cdot,\cdot)}\Rightarrow 1] \tag{2.10}$$

An adversary attacking the IV-based encryption scheme $\mathcal{E}$ is said to be *nonce-respecting* if it never repeats any component of any IV-value N. In particular, if IV values are strings, then a nonce-respecting adversary never repeats an IV; if IV values are vectors of string, then a nonce-respecting adversary is obliged to do something more, to never repeats any component of any IV. Informally, an IV-based encryption scheme is secure in the IND$-sense (note the change in capitalization) if $\mathbf{Adv}^{\mathrm{IND\$}}_{\mathcal{E}}(\mathcal{A})$ is "small" whenever $\mathcal{A}$ is "reasonable" and nonce-respecting.

The IND$-security definition is one particular SemCPA notion. All the same weakenings of ind$-security apply to IND$-security, too, giving other SemCPA security notions and analogous results to those in the probabilistic setting; the adjustment in notions from probabilistic to nonce-based makes no significant change in the landscape of the security notions nor the implications among them. Just as before, showing IND$-security is the "best" one can do for establishing positive results; the notion would seem to be the strongest SemCPA security notion in the literature.

I.8. Nonce-based security is preferred. I have heard it claimed that the probabilistic and the nonce-based notions of security for symmetric encryption are incomparable; neither notion is "better" or "stronger" than the other, they are simply different. This statement is technically true when the two kinds of objects are formalized using distinct syntax, as has always been done

in the past, although, at a "meta-level," the statement feels false. Now that we have adopted a common syntax for probabilistic and nonce-based schemes, it is no longer possible to claim that probabilistic security is simply different from, not weaker than, nonce-based security.

Let $\mathcal{E}\colon \mathcal{K} \times \{0,1\}^n \times \mathcal{X} \to \mathcal{X}$ be an IV-based encryption scheme. Then if it is "good" in the nonce-based framework—the IND$-definition—then it is also good in the probabilistic setting—the ind$-definition. This is fairly obvious: apart from a $q^2/2^{n+1}$ term to account for the collision of randomly generated IVs, the IVs that are randomly generated by the experiment defining security in the probabilistic setting will generate IVs that qualify as nonces, and so one will suffer a quantitative security degradation of just $q^2/2^{n+1}$. It is also obvious that the converse claim is *not* true: a scheme can work well for random IVs, but fail completely when, for example, a counter-realized nonce is used instead. In fact, CBC, CFB, and OFB are all this way, as we soon shall see.

It is strongly preferred for an IV-based encryption scheme to be secure with respect to the nonce-based notion, not just the probabilistic one. The reason is that a nonce-secure scheme is less likely to be misused. Good approximations to randomness can be hard to achieve in the highly deterministic world of digital computers, whereas statefulness—something like a counter—is usually more easily achieved. In fact, in contemporary security practice good approximations to randomness are usually achieved with the help of state, employing a process in which a pool of unpredictable bits is harvested and then this is processed, by a stateful process, to produce a stream of pseudorandom bits. A failure to provide a good approximation to randomness is at the heart of many cryptographic vulnerabilities.

It is not hard to give CBC, CFB, and OFB variants that are secure in the nonce-based sense—in particular, it *always* works to take a probabilistic scheme $\mathcal{E}$ and convert it to a nonce-based one $\widehat{\mathcal{E}}$ by asserting that $\widehat{\mathcal{E}}^N_{K\,K'}(P) = \mathcal{E}^R_K(P)$ where $R = E_{K'}(N)$. See [180] for the case of CBC. Indeed NIST suggests doing this, except that they set $K = K'$, which does not quite work with respect to our definitions.

I.9. Security of a tweakable-blockcipher. The final notion of security for an IV-based encryption is security as a good tweakable blockcipher. We will defer a discussion of tweakable blockciphers to Chapter 6, when the idea will be needed. Here we simply note two things. First we comment that tweakable blockciphers don't have to have a "short" blocklength, like 128 bits; as with blockciphers, we consider that possibility that tweakable blockciphers may take in long strings, including strings of various lengths. Second, we comment that, having put tweakable blockciphers on the same syntactic footing as probabilistic and nonce-based encryption schemes, one can now compare all of these security notions. It it is easy to see that a "good" tweakable blockcipher—good in the sense of a tweakable PRP, as we will describe—implies being a good nonce-based encryption scheme (which, in turn, implies being a good probabilistic encryption schemes). None of the reverse implications are valid.

I.10. The danger of privacy-only schemes. Fundamentally, *all* confidentiality-only modes—especially CBC mode, since users seem to often assume otherwise—are highly vulnerable to misuse. At issue is the fact that, at best, the modes *only* provide for SemCPA security, which isn't as strong as it sounds. As soon as one steps out of the SemCPA setting, even in a seemingly "innocent" way, all bets are off.

A striking example of the limitations of SemCPA security is what happens if mandatory padding is incorporated into CBC mode, as suggested in [61, Appendix A], and an oracle is provided to the adversary, in addition to the usual one, that tells it whether or not the provided ciphertext is the encryption of a correctly padded plaintext. Starting with Vaudenay [194] and

continuing with many follow-on works [42, 49, 165, 207], it has been convincingly demonstrated that, for many padding schemes and modes, this is quite enough to destroy a mode's security, often permitting efficient decryption of arbitrary texts. (Note that, while the attack model does go beyond indistinguishability under a chosen-plaintext attack, it still falls far sort of permitting a chosen-ciphertext attack.)

The fact that a SemCPA-secure scheme like CBC can be utterly destroyed by something so "innocent" and often feasible to realize suggests, to many, that SemCPA notions are too weak, and that modes that only target such a goal are fundamentally dangerous. Weak security notions are inimical to ease of correct use.

The weakness of privacy-only schemes is not a "theoretical" problem. In practical settings, the deficiencies often matter. As a striking example, Degabriele and Paterson [56], following Bellovin [28] and Paterson and Yau [166], detail fatal problems for privacy-only IPsec configurations. Such IPsec configuration are highly insecure even if the defining RFCs are perfectly adhered to. The problem is, at heart, that SemCPA security—both as a notion and as a set of commonly-used schemes—is just not rich enough to model attacks that arise when the schemes are embedded in practical, real-world protocols.

Black and Urtubia convincingly argue that attacks like Vaudenay's should be interpreted as indicating that our blockcipher modes of operation ought to be providing a service that goes well beyond what SemCPA security is guaranteed to do: good modes should provide for *authenticated encryption*, the authors maintain. They write:

> Authentication has long been thought of as a separate security goal [from privacy], used only when one is concerned about such things as message integrity, authenticity of origin, and non-repudiation. It is often regarded as unnecessary in systems which require only privacy. But as we have seen, the ability to freely manipulate ciphertexts gives rise to a wide range of possible attacks based on side-channels which presumably cannot be exploited without this ability. These side-channels are demonstrably damaging and it is likely very difficult to avoid constructing them in real systems. In light of this, perhaps it is time to view authentication as a strongly-desirable property of any symmetric encryption scheme, including those where privacy is the only security goal. [42, p. 336]

While I am loathe to recommend that *no* privacy-only mode be included in contemporary cryptographic standards, I am not far from holding that opinion. By now, it should be well recognized that privacy-only encryption modes are certain to be extensively *mis*used if they are extensively used at all.

Chapter 3

ECB Mode

3.1. Summary. This chapter looks at the Electronic Codebook (ECB) mode of operation, as defined in NIST Recommendation SP 800-38A [61]. ECB is one of the four "classical" modes of operation defined for DES more than 30 years ago, in FIPS 81 [153]. But classicism does not correctness make; the mode has some fundamental and well-known problems. First among these is that identical blocks of plaintext yield identical corresponding blocks of ciphertext. This characteristic is already enough to imply that ECB will not achieve any desirable privacy property that we know.

Now as a deterministic mode—no initialization vector, randomness, or state—ECB never had any chance to achieve something like SemCPA security; at the very least, repetitions of plaintexts must, in a deterministic scheme, be revealed as repetitions in the corresponding ciphertexts. But what ECB "leaks" seems to run much deeper. Syntactically, ECB must be considered as a blockcipher—a variable-input-length (VIL) one. But ECB does not do what we usually imagine that a blockcipher "ought" to do—at least not if one takes the definition of PRP-security as a normative goal. Nor does it seem to achieve any interesting relaxation of this goal, like being an on-line cipher [12], that we know.

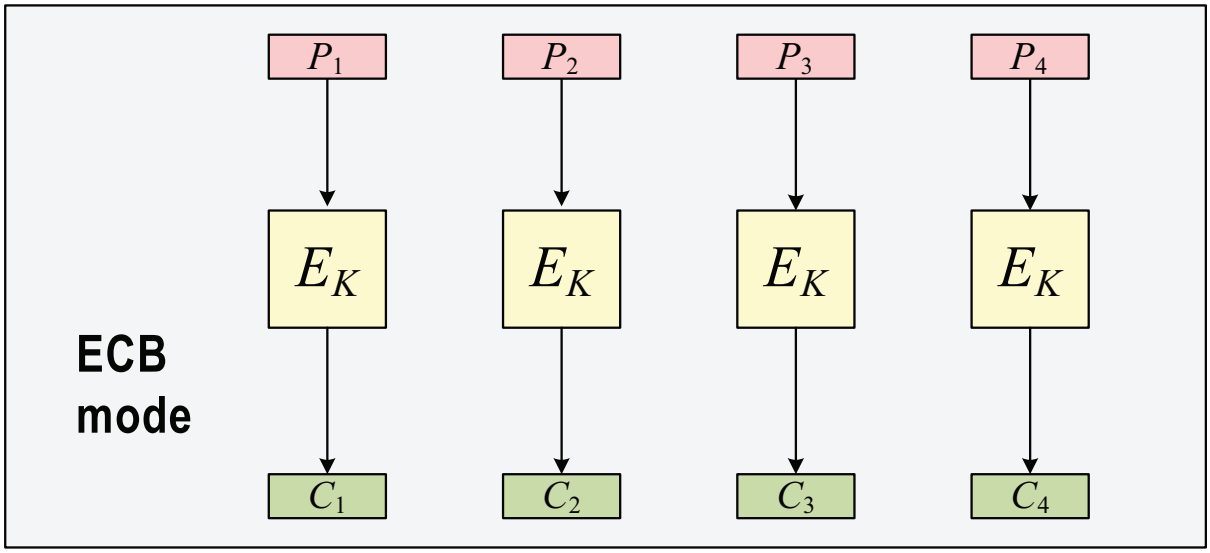
Because of its obvious leakage of potentially important information, ECB has never been regarded as "interesting" scheme by serious cryptographers; the mode was simply dismissed as being wrong. The book of Menezes, van Oorschot, and Vanstone says of ECB that

> Since ciphertext blocks are independent, malicious substitution of ECB blocks (e.g., insertion of a frequently occurring block) does not affect the decryption of adjacent blocks. Furthermore, block ciphers do not hide data patterns—identical ciphertext blocks imply identical plaintext blocks. For this reason, the ECB mode is *not recommended for messages longer than one block, or if keys are reused for more than a single one-block message.* ([139, Remark 7.12], emphasis my own.)

A mode that should only be used once per key, and on a message of just one block, would seem to be of insignificant utility.

Just the same, ECB has persisted for decades, getting replicated, (apart from changes like allowing non-DES blockciphers) in multiple standards, including ANSI X3.106, ISO 8732, and ISO/IEC 10116. It must do *something* of value, no? One might note:

- If plaintext are *uniformly random* and are multiples of n-bits (the block size of the block cipher), than ECB will work just fine to provide confidentiality. In the context of transporting a session key, this scenario is common. Privacy with respect to a "random"



10 **algorithm** $\mathrm{ECB}_K(P)$ *ECB mode*
11 // $K \in \mathcal{K}$, $P \in (\{0,1\}^n)^+$
12 $P_1 \cdots P_m \leftarrow P$ where $|P_i| = n$
13 **for** $i \leftarrow 1$ **to** m **do** $C_i \leftarrow E_K(P_i)$
14 $C \leftarrow C_1 \cdots C_m$
15 **return** C

Figure 3.1: **ECB mode.** Only encryption is shown; decryption is the unique corresponding inverse. The scheme is parameterized by a blockcipher $E\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ and each block of the input is separately processed by E. The mode is not defined when the input string has length that is a non-multiple of n.

message attack is an easy aim to formalize and prove that ECB enjoys.

- Criticisms that effectively speak to the absence of authenticity or integrity, or speak to the malleability of ciphertexts, are largely off the mark, since none of the standard confidentiality modes offer such protection. The first sentence from the quote above by Menezes, van Oorschot, and Vanstone is infused with such concern.
- ECB makes a powerful building block for *other* schemes. For example, CTR mode (Chapter 5) may be regarded as being "built" from ECB. The same can be said of OCB mode [182], and Naor and Reingold's NR-mode [150, 151]. ECB offers useful descriptive language for describing more complex cryptographic modes.
- In cases like those just named, it is not *only* the descriptive value that ECB brings to the table: implementations are often designed to fall along this abstraction boundary. As an example, cryptographic hardware and cryptographic libraries often export an ECB interface/API, and this interface can and is used to implement useful modes on top of ECB. The ECB mode in such cases serves as an clean architectural boundary rather than as a confidentiality mode in its own right.

Given the considerations just named, and given the extensive history of ECB, I would not be opposed to including ECB in a portfolio of useful and important modes. At the same time, ECB should not be regarded as a user-ready, general-purpose scheme.

In §3.3 we sketch an answer the following question: what privacy property *does* ECB achieve? The answer we seek should fall along the following lines. We know that ECB leaks *something*,

namely, the equality of message blocks across distinct positions and time. We would like to formalize this particular something, defining a "leakage function" $\mathcal{L}_{\text{ECB}}$ corresponding to it. We call this leakage function the *block-repetition* leakage function. We define what it means to leak only $\mathcal{L}_{\text{ECB}}$ (and, more generally, we define what it means to leak only $\mathcal{L}$). The notion is new.

Later, when we get to XTS mode in Chapter 6, we will lift the security-up-to-$\mathcal{L}$ notion to that setting, too.

The idea of being private up to some function $\mathcal{L}$ lifts to other settings. In particular, one could speak of being nonmalleable up to some function $\mathcal{L}$. One would expect that ECB does achieve some kind of nonmalleability up to $\mathcal{L}_{\text{ECB}}$. The approach may prove to be useful for comparing or speaking about "how nonmalleable" or "how close to private" a flawed scheme may be.

3.2. Definition of the scheme. The ECB mode of operation is defined and illustrated in Figure 3.1. It depends on a blockcipher $E\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$. The NIST Recommendation asserts that the blockcipher must be FIPS-approved. At present there are three FIPS-approved blockciphers: AES [154], TDEA [10], and Skipjack [156, 160]. The most popular of these, AES, comes in three key lengths.

3.3. Leaking "only" block repetitions. Recall Goldwasser and Micali's (GM's) English-language description of semantic security [74]: that which an adversary can compute about the plaintext given the ciphertext it can also compute about the plaintext in the *absence* of the ciphertext. We wish to refine this idea by saying that a scheme is semantically secure "up to $\mathcal{L}$" if that which an adversary can compute about the plaintext given the ciphertext it can *also* compute about the plaintext given *just* the leaked information about the ciphertext, $\mathcal{L}(C)$. The function $\mathcal{L}$ is what is *understood* to be leaked. It may be information that we don't *mind* relinquishing.

Despite the English-language description suggesting that the adversary in GM's notion is to be given *nothing* about the ciphertext, the formalization actually *does* give the adversary something important about the ciphertext: its *length*. We wish to recover the symmetric-setting notion of semantic security, or one of its equivalent formulations [14], not when the leakage function is null, but when we leak just the message length, $\mathcal{L}(C) = |C|$.

Now getting a bit more formal, we imagine a function $\mathcal{L}$, the *leakage function*. Each invocation of $\mathcal{L}$ takes in a ciphertext $C \in \{0,1\}^*$ and the current state s. The function outputs a string λ (the information that is being leaked) and a modified state s', namely, $(\lambda, s') \leftarrow \mathcal{L}(C, s)$. We will henceforth omit s and s' from the notation, writing $\lambda \leftarrow \mathcal{L}(s)$ but understanding that, implicitly, state s is updated during the call. Note that the state s may "grow" during a sequence of calls to $\mathcal{L}$, and that the saved state s following a sequence of calls $\mathcal{L}(C_1), \ldots, \mathcal{L}(C_t)$ may encode $C_1, \ldots, C_t$.

The intent is that $\mathcal{L}(C)$ encodes information that the adversary inherently knows about P given C and given the sequence of prior ciphertexts $C_1, \ldots, C_t$. In the case of ECB, $\mathcal{L}(C)$ needs to encode information like

> Block-3 and block-5 of the plaintext corresponding to C are identical; block-2 of its plaintext is the same as block-1 of the plaintext corresponding to the *prior* ciphertext; and all other plaintext blocks are different from each other, different from the two plaintext blocks we have just mentioned, and different from all prior plaintext blocks.

Quite a mouthful.

```
20  algorithm L_ECB(C)                                   Leakage function for ECB
21  C_1 ··· C_m ← C where |C_i| = n
22  for i ← 1 to m do
23      if T[C_i] = undefined then T[C_i] ← Cnt;  Cnt ← Cnt + 1
24      λ_i ← T[C_i]
25  λ ← ⟨λ_1, λ_2, ··· , λ_m⟩
26  return λ
```

Figure 3.2: **Leakage function for ECB.** The program formalizes that which we *know* leaks from ECB mode. Given an arbitrary leakage function $\mathcal{L}$, we can formalize what it means to leak *only* it.

To encode such information concisely, number the distinct blocks the adversary sees, in order, by $0, 1, 2, \ldots$. Provide, in response to an m-block query C, the vector of numbers that describes which numbered blocks appeared where. See Figure 3.2. For the pseudocode there we implicitly initialize the associative array T to everywhere undefined, and the counter Cnt to 0. These variables are static; they comprise the state s of which we earlier spoke. Leakage function $\mathcal{L}_{\text{ECB}}$ for ECB implicitly depends on the block size n that will be used with ECB.

Having defined the information $\mathcal{L}$ about a ciphertext that we *know* ECB will leak, we must define confidentiality up to this. We spoke earlier of semantic security, but let us first describe an indistinguishability notion, which is simpler to deal with. Fix a blockcipher (possibly a VIL one like ECB[E]) $\mathcal{E}\colon \mathcal{K} \times \mathcal{X} \to \mathcal{X}$. Now we define the behavior of two different oracles:

- Oracle **REAL** begins its life by selecting a random $K \xleftarrow{\$} \mathcal{K}$. After that, on input of the encoding of a sampling algorithm $\mathcal{P}$ (which probabilistically outputs a point in $\mathcal{P}$), the oracle selects $P \xleftarrow{\$} \mathcal{P}$, computes $C \leftarrow \mathcal{E}_K(P)$, and returns C. To be explicit in indicating the oracle's dependencies we would write $\mathbf{REAL}_{\mathcal{E}}(\cdot)$.
- Oracle **FAKE** begins its life by selecting a random $K \xleftarrow{\$} \mathcal{K}$. After that, on input of the encoding of a sampling algorithm $\mathcal{P}$ as above, the oracle selects $P \xleftarrow{\$} \mathcal{P}$, computes $C \leftarrow \mathcal{E}_K(P)$, and then computes $\lambda \leftarrow \mathcal{L}(C)$. We now feed λ to a (possibly stateful and probabilistic) *simulator*, S, obtaining a string $C' \xleftarrow{\$} S(\lambda)$. We return C'. Explicitly indicating the oracle's dependencies we'd write $\mathbf{FAKE}_{\mathcal{E},\mathcal{L},S}(\cdot)$. The simulator S may be stateful in the same sense that $\mathcal{L}$ may be; with each call, it implicitly updates its own saved state.

We note that oracle **REAL** can do everything an encryption oracle $\mathcal{E}_K$ can do: if the adversary gives it the description of a sampling algorithm that (always) returns some particular point P, then the oracle will return $\mathcal{E}_K(P)$. But the oracle can do more, in the sense that it can model the return of ciphertext on which the adversary does *not* know everything about the corresponding plaintext.

The point to note is that, regardless of what the simulator S does, regardless of how long it computes, the (fake) "ciphertext" it produces does not depend on P, except so far as P influences $\lambda = \mathcal{L}(\mathcal{E}_K(P))$, the information about P that we *know* will be leaked in its encryption. So if there is a simulator S that makes oracles **REAL** and **FAKE** indistinguishable, it means that what the adversary sees in a (real) ciphertext amounts to no more than what it could get on its own knowing *just* the information λ. We thus define

$$\mathbf{Adv}^{\text{IND}[\mathcal{L},S]}_{\mathcal{E}}(\mathcal{A}) \quad = \quad \Pr[\mathcal{A}^{\mathbf{REAL}_{\mathcal{E}}(\cdot)} \Rightarrow 1] - \Pr[\mathcal{A}^{\mathbf{FAKE}_{\mathcal{E},\mathcal{L},S}(\cdot)} \Rightarrow 1]\,. \tag{3.1}$$

To show a scheme private-up-to-$\mathcal{L}$ one would exhibit a (preferably simple and efficient) simulator S such that $\mathbf{Adv}^{\text{IND}[\mathcal{L},S]}_{\mathcal{E}}(\mathcal{A})$ is "small" regardless of what any "reasonable" adversary $\mathcal{A}$

may do.

Only small changes are needed for a notion of semantically-secure-up-to-$\mathcal{L}$. This time, the adversary, after speaking to either **REAL** or **FAKE**, resulting in some vector of underlying messages (one for each query) $P = (P_1, \ldots, P_q)$, outputs a pair (f, y), the encoding of a function and a string, an assertion that the adversary "thinks" that $f(P) = y$. We compare the adversary's chance of being right about this assertion in the case where real ciphertexts have been issued and the case where fake ciphertexts have been issued, again by some simulator S that knows only the known-to-be-leaked information. We consider the event that

$$\begin{aligned}\mathbf{Adv}^{\mathrm{sem}[\mathcal{L},S]}_{\mathcal{E}}(\mathcal{A}) \;&=\; \Pr[(f,y) \leftarrow \mathcal{A}^{\mathbf{REAL}_{\mathcal{E}}(\cdot)} :\; f(P) = y] - \\ &\qquad \Pr[(f,y) \leftarrow \mathcal{A}^{\mathbf{FAKE}_{\mathcal{E},\mathcal{L},S}(\cdot)} :\; f(P) = y] \end{aligned} \tag{3.2}$$

where P, in each experiment, is the vector of internally selected plaintexts. To show a scheme semantically-secure-up-to-$\mathcal{L}$ one would exhibit a (preferably simple and efficient) simulator S such that $\mathbf{Adv}^{\mathrm{sem}[\mathcal{L},S]}_{\mathcal{E}}(\mathcal{A})$ is small regardless of what the adversary $\mathcal{A}$ may do.

Let us sketch the (easy) result that that ECB is private-up-to-$\mathcal{L}$ for the leakage function $\mathcal{L}$ of Figure 3.2. One the i-th query (remember that the simulator S is allowed to be stateful) it is passed some a vector $\langle \lambda_1, \ldots, \lambda_m \rangle$. The simulator itself maintains a list $T = (T_0, T_1, T_2, \ldots)$. For i running from 1 to m the simulator looks to see if there is already a λ_i-indexed element of T. If there is not, then simulator creates one, giving it a uniform random value in $\{0,1\}^n \setminus T$ (here regarding the elements in T as a set instead of as a sequence). When finished, simulator S returns the (fake) ciphertext $C \leftarrow T_{\lambda_1} \,\|\, T_{\lambda_2} \,\|\, \cdots \,\|\, T_{\lambda_m}$.

It is easy to see that the view imparted by simulator S in game **FAKE** is precisely the view imparted by game **REAL** when the encryption scheme is ECB taken over a random permutation. As a consequence, we have our adversary $\mathcal{B}$ for breaking the PRP-ness of E that achieves advantage $\mathbf{Adv}^{\mathrm{IND}[\mathcal{L},S]}_{\mathcal{E}}(\mathcal{A})$. The simulation being efficient, the running time of $\mathcal{B}$ will be about that of $\mathcal{A}$, and the number of queries asked by $\mathcal{B}$ will be the total number of blocks asked by $\mathcal{A}$.

* * *

One unpleasant characteristic of the definition we have given is the presence of the simulator S. It can be banished in an alternative formulation where one replaces the leakage function $\mathcal{L}$ with a *filter* $\mathcal{F}$. This map, once again implicitly stateful, takes in a ciphertext C and returns a "canonical plaintext" $P = \mathcal{F}(C)$ that again captures that which an adversary *will* know about the plaintext, but here assumed to take the form of something encodable by a point in the message space. Use of a filter $\mathcal{F}$ amounts to an assertion that a ciphertext C resembles its "real" plaintext $P = \mathcal{D}_K(C)$ no more than it resembles the canonical plaintext $P' = \mathcal{F}(C)$. As before, let us provide an indistinguishability-style definition, but now without the simulator. Fix a deterministic encryption scheme $\mathcal{E} :\; \mathcal{K} \times \mathcal{X} \to \mathcal{X}$. Now we define the behavior of two different oracles:

- Oracle **Real** begins its life by selecting a random $K \xleftarrow{\$} \mathcal{K}$. After that, on input of a message $P \in \mathcal{X}$ the oracle computes $C \leftarrow \mathcal{E}_K(P)$, and then returns C. To be explicit in indicating the oracle's dependencies we would write $\mathbf{Real}_{\mathcal{E}}(\cdot)$.

- Oracle **Fake** also begins its life by selecting a random $K \xleftarrow{\$} \mathcal{K}$. After that, on input of a message $P \in \mathcal{X}$ the oracle computes $C \leftarrow \mathcal{E}_K(P)$, $P' \leftarrow \mathcal{F}(C)$, $C' \leftarrow \mathcal{E}_K(P')$, and then returns C'. Explicitly indicating the oracle's dependencies we'd write $\mathbf{Fake}_{\mathcal{E},\mathcal{F}}(\cdot)$.

```
30  algorithm F_ECB(C)                                          Filter function for ECB
31  C_1 ··· C_m ← C where |C_i| = n
32  for i ← 1 to m do
33      if T[C_i] = undefined then T[C_i] ← Cnt;  Cnt ← Cnt + 1
34      λ_i ← T[C_i]
35  P' ← [λ_1]_n ‖ [λ_2]_n ‖ ··· ‖ [λ_m]_n
36  return P'
```

Figure 3.3: **Filter for ECB mode.** The (stateful) program again formalizes that which we *know* leaks from ECB mode, but now by way of returning a canonical plaintext for each ciphertext, the plaintext capturing the information that can be considered known and implicit when the ciphertext is given.

The filter $\mathcal{F}_{\mathrm{ECB}}$ for ECB mode is defined in Figure 3.3. It is just like the leakage function $\mathcal{L}_{\mathrm{ECB}}$ except that the integer labels $0, 1, 2, \ldots$ that used to be returned are replaced by corresponding n-bit strings (recall that $[\, i \,]_n$ is the encoding of the number i and an n-bit string). We define

$$\mathbf{Adv}^{\mathrm{IND}[\mathcal{F}]}_{\mathcal{E}}(\mathcal{A}) \quad = \quad \Pr[\mathcal{A}^{\mathbf{Real}_{\mathcal{E}}(\cdot)} \Rightarrow 1] - \Pr[\mathcal{A}^{\mathbf{Fake}_{\mathcal{E},\mathcal{F}}(\cdot)} \Rightarrow 1] \; . \tag{3.3}$$

Only small changes are needed for a notion of semantically-secure-up-to-$\mathcal{F}$.

Once again it is straightforward to show that ECB mode is secure up to the filter $\mathcal{F}_{\mathrm{ECB}}$ that we defined. While slightly less natural, the filter-function approach seems technically cleaner than the leakage-function one. We prefer this notion, only defining the simulation-based approach because this seemed a good pedagogic step.

Summarizing, when we say that a blockcipher $\mathcal{E}$ is secure up to the function $\mathcal{F}$ we are asserting that what the adversary learns from possession of an $\mathcal{E}_K(\cdot)$ oracle is indistinguishable from that which it would learn from a $\mathcal{E}_K(\mathcal{F}(\mathcal{E}_K(\cdot)))$. To recover a conventional PRP notion of blockcipher security, one would have filter $\mathcal{F}$ "mark" repetitions of (entire) ciphertexts. To formalize what ECB leaks, the filter marks repetitions of ciphertext blocks. Leaving deterministic encryption schemes behind, a SemCPA notion of security would be recovered with a filter function $\mathcal{F}$ that returns, for any ciphertext C, the plaintext 0^m where m is the length of the decryption of C, assumed to be computable from C.

3.4. Comments. It is fair to ask if we have *done* anything in §3.3: the notion we built up to is of unclear utility. The proof that ECB achieves the notion would be straightforward. Nonetheless, I would suggest that it may be nice to have a formalized language that lets one explicitly speak to just *what* is leaked in an enciphering scheme. It lets one say something, compactly and rigorously, about ECB; it lets one rigorously compactly distinguish *what* is leaked in ECB from what is leaked in a tweakable version of ECB [124], an on-line cipher [12], or a PRP-secure blockcipher like EME [83]. It lets one say just *what* is leaked in each case, and emphasizes that, even in the PRP setting, what is leaked is still not nothing. The viewpoint bolsters a point of view that is not exactly that ECB is "wrong" which is the conventional, prescriptive, and perhaps dogmatic view of matters. It is more that the mode leaks quite a large amount of information—that which is captured by the filter function $\mathcal{F}_{\mathrm{ECB}}$. In a given application, leaking this information may or may not be an acceptable thing to leak.

Chapter 4

CBC, CFB, and OFB Modes

4.1. Summary. This chapter looks at the three classical IV-based modes of operation: CBC (Cipher Block Chaining), CFB (Cipher Feedback), and OFB (Output Feedback). The modes were originally defined in FIPS 81 [153]. At that time the techniques were specific to DES, and OFB was defined more generally that it is in SP 800-38A [61]. Along with ECB, these three modes of operation have a distinguished status springing from their age.

We lump together CBC, CFB, and OFB because they share some basic characteristics. All are secure, in the SemCPA sense, if the user employs a random IV. On the other hand, none of the modes are secure, in the SemCPA sense, if one merely uses a nonce IV. All of the mechanisms are dated, employing chaining for purposes unconvincing from a modern point of view. Overall, it seems hard to find any good reason why one would prefer either CBC or OFB over CTR, while the main characteristic of CFB mode that would usually underlie its selection—self synchronization—has been shown to be achievable at far lower cost by a (not standardized) alternative, OCFB, described by Alkassar, Geraldy, Pfitzmann, and Sadeghi [2].

We treat ECB separately from CBC/CFB/OFB because it is not IV-based and does not enjoy SemCPA security. We treat CTR separately because it achieves a nonce-based notion of security, its IV is not an n-bit string, and its historical context is different. We treat XTS separately because it enjoys no sort of SemCPA security and its historical context is again different from the rest.

We summarize the basic characteristics of CBC, CFB, and OFB in Figure 4.1. The table serves as a concise distillation of much of what is said in this chapter.

Despite my rather negative appraisal of the CBC, CFB, and OFB modes, I fall short of saying that these modes should not be included in a portfolio of modes. The main reason for this is that widespread use can itself be an argument for standardization, and it is clear that CBC, especially, is very widely used. Also, CFB does have an interesting property—self synchronization—that may be important in some real-world applications and that is not shared with any other standardized mode. All of that said, I am unable to think of any cryptographic design problem where, absent major legacy considerations, any of CBC, CFB, or OFB would represent the mode of choice.

4.2. Definitions of the schemes. We define and illustrate the CBC, CFB, and OFB modes of operation in Figures 4.2–4.4.

- CBC is shown in Figure 4.2. Its only parameter is a blockcipher $E\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$. The message space for CBC is $\mathcal{X} = (\{0,1\}^n)^+$; in other words, plaintexts must be a positive

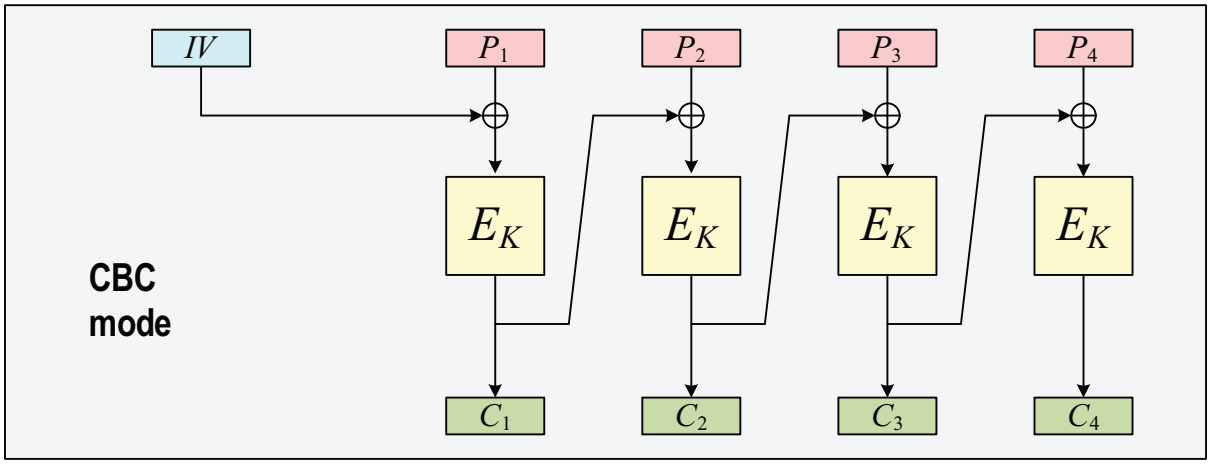
Characteristic	CBC	CFB	OFB
SemCPA secure if the IV is **random**.	**Yes**. Proof in [14]. The ind$ security notion is easy to establish.	**Yes**. Proof in [2]. The ind$ security notion can also be shown.	**Yes**. Follows directly from ind$-security of CBC.
SemCPA secure if the IV is a **nonce**.	**No**. Enciphering nonce under E_K also incorrect. Incorrect use is common.	**No**. Enciphering nonce under E_K also incorrect.	**No**. But secure with a fixed sequences of IVs, like a counter.
SemCCA secure.	**No**.	**No**.	**No**.
Nonmalleable (cannot manipulate ciphertexts to create related plaintexts)	**No**. Often wrongly assumed to provide some sort of nonmalleability.	**No**. Trivially malleable.	**No**. Trivially malleable.
Padding oracle (says if the ciphertext is correctly padded) is well tolerated.	**No**. Pad oracle destroys SemCPA for most pad methods [42, 165, 194].	Usually **Yes** (or **NA**) as no padding is needed for small enough s.	Usually **Yes** (or **NA**) as no padding is ever needed with OFB.
Error propagation: will recover if a ciphertext bit is flipped (a "bit error").	**Yes**. Changes to C_i affect two blocks, P_i and P_{i+1}	**Yes**. Changes to C_i affect about n/s data segments	**Yes**. Changes to C_i affect only P_i
Self-synchronous: will recover if insert / delete ciphertext bits (a "slip").	**No**, unless the slip is a multiple of n-bits.	**Yes** for slips of s-bits (so always if $s = 1$). **No** for other slips.	**No**.
Maximal rate scheme: processes n-bits for every blockcipher call.	**Yes** (n bits processed per blockcipher call).	**No** (when $s < n$). The mode processes s bits per blockcipher call.	**Yes** (n bits processed per blockcipher call).
Parallelizable with respect to **encryption**.	**No**. Can always use interleaving to enhance parallelizability.	**Yes**, but awkward. See ISO 10116 for less awkward generalization.	**No**.
Parallelizable with respect to **decryption**.	**Yes**. Block P_i depends only on C_i, C_{i-1}.	**Yes**, but awkward.	**No**.
Inverse-free (blockcipher inverse is unnecessary for decryption).	**No**.	**Yes**.	**Yes**.
Preprocessing improves encrypt/decrypt speed.	**No**.	**No**.	**Yes**.

Figure 4.1: **Summary characteristics of three IV-based encryption schemes.** We assume an n-bit blockcipher and, for CFB mode, an s-bit data segment. We group together security properties (first), synchronization and error-propagation characteristics (second), and efficiency characteristics (third). The wording of mode characteristics has been selected so that a "yes" would generally be considered a "desirable" characteristic.

multiple of n bits. The IV space is $\mathcal{IV} = \{0,1\}^n$.

- CFB is shown in Figure 4.3. This time the mode has two parameters: a blockcipher $E\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ and a *data-segment length*, s, where $1 \le s \le n$. The message space for CBC is then $\mathcal{X} = (\{0,1\}^s)^+$; plaintexts must be a multiple of s bits. Typical values for s are $s = 1$, $s = 8$, and $s = n$. The IV space is $\mathcal{IV} = \{0,1\}^n$.
- OFB is shown in Figure 4.4. The only parameter for the mode of operation is a blockcipher $E\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$. The message space for OFB is $\mathcal{X} = \{0,1\}^*$; plaintexts of any length may be encrypted. The IV space is $\mathcal{IV} = \{0,1\}^n$.

For all of the modes, we define only the encryption direction of the scheme. As explained already in §I.5, this is sufficient in the sense that the corresponding decryption algorithm is well defined once the encryption direction $\mathcal{E}_K^{IV}(\cdot)$ is specified.



10 **algorithm** $\mathrm{CBC}_K^{IV}(P)$ *CBC mode*
11 // $K \in \mathcal{K}$, $IV \in \{0,1\}^n$, $P \in (\{0,1\}^n)^+$
12 $P_1 \cdots P_m \leftarrow P$ where $|P_i| = n$
13 $C_0 \leftarrow IV$
14 **for** $i \leftarrow 1$ **to** m **do** $C_i \leftarrow E_K(P_i \oplus C_{i-1})$
15 $C \leftarrow C_1 \cdots C_m$
16 **return** C

Figure 4.2: **CBC mode.** The scheme is parameterized by a blockcipher $E\colon\ \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$. The mode is not defined when the input string has length that is a non-multiple of n.

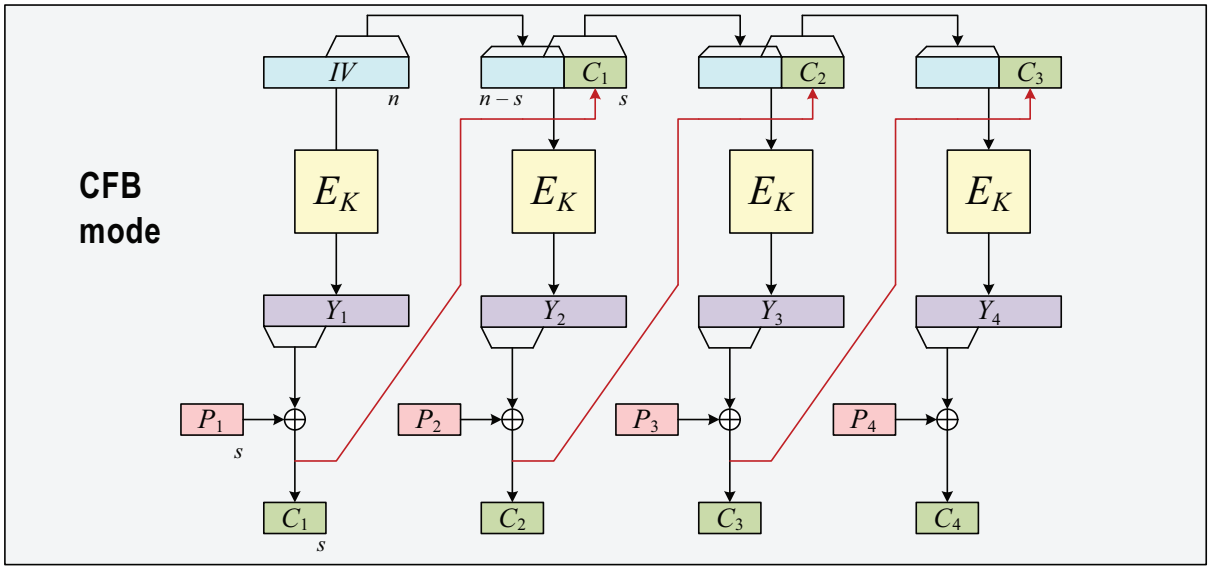
We remind the reader that, under our formulation of IV-based encryption, the IV is not included as part of the ciphertext.

As per SP 800-38A, the blockcipher E is not arbitrary; it is required to be NIST-approved, which means, at present, that the algorithm must be AES (either AES-128, AES-192, or AES-256) [158]), Skipjack [156], or TDEA (either TDEA-112 or TDEA-168, corresponding to keying options 2 and 1 of the defining spec [10]). A list of approved blockciphers is maintained by NIST [155].

The hardest to understand of the modes is CFB. It can be explained as follows. First we initialize a *feedback buffer* named X to the initialization vector IV. Then we process each s-bit data segment P_i as follows: (1) apply E_K to X of get the blockcipher output Y; (2) use the first s bits of Y to Vernam-style encrypt P_i and output the corresponding ciphertext C_i; (3) now adjust the feedback buffer X by shifting off the leftmost s bits (the bits that were just used to mask P_i), shifting in (as s new bits) the ciphertext C_i just produced. Because the contents of feedback buffer X are shifted (by s bits) with each data segment processed, the feedback buffer is often implemented by or regarded as a shift register.

4.3. Versions of CBC, CFB, and OFB. Modes CBC, CFB, and OFB have all been defined differently—either in greater or in lesser generality—in alternative standards documents (that is, alternatives to SP 800-38A [61]). These alternative standards include ANSI X9.52 [5], ISO/IEC 10016:2006 [92], and the "original" modes of operation specification, FIPS 81 [153]. Focusing on the most recent of these standards, ISO 10016:2006, we find several important differences:

- The ISO standard allows "interleaving" for CBC. One selects a "stride length" $m \geq 1$ and separately applies m instances of CBC encryption, each application to blocks that are separated by spans of $m-1$ intervening blocks. One now needs m separate IVs for mode.



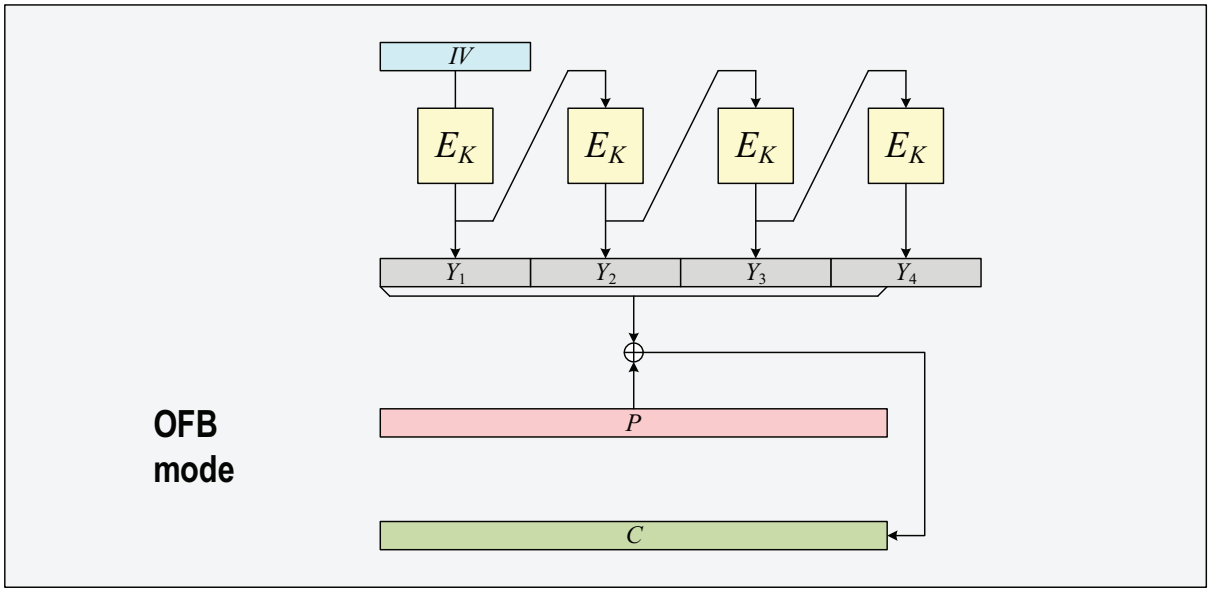
20 **algorithm** $\mathrm{CFB}_K^{IV}(P)$ *CFB mode*
21 // $K \in \mathcal{K}$, $IV \in \{0,1\}^n$, $P \in (\{0,1\}^s)^+$
22 $P_1 \cdots P_m \leftarrow P$ where $|P_i| = s$
23 $X_1 \leftarrow IV$
24 **for** $i \leftarrow 1$ **to** m **do**
25 $\quad Y_i \leftarrow E_K(X_i)$
26 $\quad C_i \leftarrow P_i \oplus \mathrm{MSB}_s(Y_i)$
27 $\quad X_{i+1} \leftarrow \mathrm{LSB}_{n-s}(X_i) \,\|\, C_i$
28 $C \leftarrow C_1 \cdots C_m$
29 **return** C

Figure 4.3: **CFB mode.** The scheme is parameterized by a blockcipher $E\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$. The mode is not defined when the input string has length that is a non-multiple of n.

The method may be considered a folklore approach for getting a measure of parallelizability. The "poor-man's parallelizability" one gets from interleaving CBC would work just as well for interleaving other modes of operation, including OFB.

– The ISO standard's version of CFB is highly general—it is general to the point of being rather difficult to understand. Besides the parameter s, two additional parameters are included: the length r of the feedback buffer, which may now exceed n; and an additional parameter k, where $s \le k \le n$. The CFB algorithm is generalized so that only the leftmost n bits of the feedback buffer are fed into the blockcipher. The feedback buffer is now updated by shifting off the leftmost k bits, rather than s, and inserting additional one-bits, if needed, prior to the s-bit ciphertext block that will be shifted in on the right. The standard suggests $s = k$ [92, Section 8.1] and claims that security is best when this common value is n [92, Section B.3.2(d)]. The justification provided for this added complexity is that the mode permits pipelining if $r \ge n + k$ [92, Section B.3.2(f)]. We comment that, while inelegant, any self-synchronous mode can already be pipelined. CFB as specified in the NIST standard corresponds to the ISO standard's definition with $r = n$ and $k = s$.

– The ISO standard's version of OFB mode, following FIPS 81 [153], has an extra parameter s. Only the leftmost s bits of each Y_i are used to mask P in forming the ciphertext C. The rate of the mode, meaning the number of plaintext or ciphertext bits processed per blockcipher call, is therefore reduced when $s < n$. The SP 800-38A scheme coincides with



30 **algorithm** $\mathrm{OFB}_K^{IV}(P)$ *OFB mode*
31 // $K \in \mathcal{K}$, $IV \in \{0,1\}^n$, $P \in \{0,1\}^*$
32 $m \leftarrow \lceil |P|/n \rceil$
33 $Y_0 \leftarrow IV$
34 **for** $i \leftarrow 1$ **to** m **do** $Y_i \leftarrow E_K(Y_{i-1})$
35 $Y \leftarrow \mathrm{MSB}_{|P|}(Y_1 \cdots Y_m)$
36 $C \leftarrow P \oplus Y$
37 **return** C

Figure 4.4: **OFB mode.** The scheme is parameterized by a blockcipher $E\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$. The mode is not defined when the input string has length that is a non-multiple of n.

the ISO 10116:2006 mechanism if $s = n$.

– Finally, the ISO standard applies to blockciphers that need not be NIST-approved.

While all of the changes mentioned act to make the ISO-versions of the various modes more general than the NIST versions, we comment that CTR mode, in contrast, is defined with considerably less generality in the ISO spec: CTR values (what we will call N_i when we get to Chapter 5) are assumed to be incremented by one (mod 2^n) with each plaintext block. The NIST standard, as we shall see, lets the vector of counter values be arbitrary, as long as all counter values are distinct.

The discussion should make clear that it can be ambiguous *what* algorithm one is actually speaking of in a discussion about CBC, CFB, OFB, or CTR. We shall always be meaning the NIST SP 800-38A versions of the modes except, when we speak of provable security, we necessarily think of the underlying blockcipher as being arbitrary.

4.4. Provable security of CBC, CFB, and OFB when the IV is random. All of these modes achieves SemCPA security when the IV is random. Indeed they all do well with respect to our (very strong) ind$ notion of security.

We begin with **CBC** mode. Its SemCPA security was first established by Bellare, Desai, Jokipii, and Rogaway [14]. Here we are in the setting of probabilistic encryption—the IV is selected uniformly at random with every message. See §I.6. The original security notion Bellare

et al. used was left-or-right indistinguishability. That security also holds in the (stronger) ind$ sense is widely known but is mentioned in not in any paper that I know. I have been teaching this result in my own cryptography class for years.

Let me specify the result more explicitly. Let E be a blockcipher and let us consider the CBC construction taken over E. The IV is random. Let $\mathcal{A}$ be an adversary attacking CBC[E]. As per §I.6, the adversary asks a sequence of queries $P_1, P_2, \ldots, P_q$, each a positive multiple of n bits. The adversary wants to guess whether the answers it receives are "real" ciphertexts, $\mathcal{C}_i = R_i \,\|\, C_i$ where R_i is random and $C_i = \mathrm{CBC}_K^{R_i}(P_i)$, or "bogus" ones, ciphertext $\mathcal{C}_i$ consists of $n + |P_i|$ random bits. Suppose that adversary $\mathcal{A}$ gets advantage $\epsilon = \mathbf{Adv}^{\mathrm{ind\$}}_{\mathrm{CBC}[E]}(\mathcal{A})$ when playing the defining game: this is the difference in the probabilities of the adversary outputting one in the two different settings. Suppose that $\mathcal{A}$ asks queries that total σ blocks (that is, $\sigma = \sum |P_i|/n$). Then, finally, we can conclude that there is an adversary $\mathcal{B}$ for attacking E, the description of $\mathcal{B}$ being simple and explicit in terms of the description of $\mathcal{A}$, and for which

$$\mathbf{Adv}^{\mathrm{prp}}_E(\mathcal{B}) \;\;\geq\;\; \epsilon - \frac{\sigma^2}{2^n}\,. \tag{4.1}$$

The running time of $\mathcal{B}$ is essentially the same as that of $\mathcal{A}$, and $\mathcal{B}$ makes σ oracle queries.

We will not prove the result above, which is quite routine. One passes to the information-theoretic setting where E is a family of random permutations; one employs the switching lemma to regard E as, instead, a family of random functions; and one does the analysis in that simplified setting, using a game-playing argument.

It took a few years until somebody—Alkassar, Geraldy, Pfitzmann, and Sadeghi—got around to proving the security of **CFB** mode [2]. Again we are in the probabilistic setting; the *IV* is random. The authors employed the left-or-right notion SemCPA security but, once again, the stronger ind$ notion would seem to be even easier to establish. The result is analogous to the CBC result; even the formula is the same, except that σ is now the number of r-bit data segments.

We emphasize that the provable security of CFB by Alkassar *et al.* [2] in only broad enough to cover the SP 800-38A formulation of the mode; it does not include the extra parameters r and k of the ISO 10116:2006 formulation [92]. Including these parameters would not be trivial, as there are clearly choices for the parameter set (k, r, s) that would be insecure; the security formula will have to be sensitive to their interplay. The absence of provable security for the ISO 10116:2006 formulation of CFB is an argument against replication of the mode with this degree of generality.

As for **OFB** mode, its ind$-security follows immediately from the ind$-security of CBC. This is because OFB can be regarded as a Vernam-style cipher made from a pad that is obtained by CBC-encrypting the appropriate length string of zero-blocks. We know from the ind$-security of CBC that all of the pads we will be generating will be indistinguishable from random bits, and so this will remain so after we xor in the plaintexts P_i. Formalizing this is easy, so one should regard OFB's ind$-security as an obvious corollary of CBC's ind$-security. Note that we see here one of the advantages of using a very strong definition of SemCPA security: had we used left-or-right security, or find-then-guess security, it would not be obvious that OFB would inherit this property from its holding on CBC.

We emphasize that all of the results above are limited in important ways: the attack model is strictly a chosen-plaintext one; the nonce must be uniform random bits; and security is "only"

to the birthday bound. While relaxations to unpredictable IVs would seem to be possible for all three modes, many other strengthenings—including chosen-ciphertext attack (CCA) security, nonmalleability, and nonce-based security—are not. The fact that results are in the probabilistic/SemCPA setting is not some triviality that is mumbled to cover the scope of a claimant's exposure; it is a crucial limitation. In any application, one must ask if the setting is actually one where ind$-security is going to be enough.

4.5. Birthday attacks. The SemCPA security bounds for CBC, CFB, and OFB with a random IV all show a security degradation of $\sigma^2/2^n$ where σ is the number of n-bit blocks (for CBC), the number of s-bit data segments (for CFB), or the number of full or fractional n-bit blocks (for OFB). It is worth pointing out that this bound is tight, a simple, folklore result. As an example for a matching attack, consider CBC. One attack would have the adversary ask a single long message of $P = 0^{n\sigma}$, getting back a ciphertext $C = C_0C_1 \dots C_\sigma$. The adversary would answer one (indicating that it believes it has a real encryption oracle) if all the C_i values are distinct; otherwise, it would answer zero. That the advantage, when $\sigma \ll 2^n$, grows with $\sigma^2/2^n$ follows from basic facts about the cycle structure of a random permutation: one expects a random permutation on n-bits to have one "giant cycle" (length about 2^{n-1}); another very long cycle (length about 2^{n-2}); and so on.

If one prefers not to ask a single long message, let the adversary split into q queries random queries of plaintexts having $m = \lceil \sigma/q \rceil$ blocks. Let the adversary infer the corresponding plaintext/ciphertext pairs (X, Y) for the underlying blockcipher. If an inconsistency is seen, this proves possession of a "random" oracle instead of a valid CBC-encryption oracle. The advantage will again be about $\sigma^2/2^n$ for $\sigma \leq 2^{n/2}$.

Equally simple attacks can be launched against CFB with $s = n$ and OFB modes. For CFB with $s \ll n$, I do not know an attack with advantage $\Omega(\sigma^2/2^n)$, nor have I seen any beyond-birthday-bound proof of security.

Security formalizations weaker than ind$-security can also be broken with $\sigma \approx 2^{n/2}$ queries. In general, unless special efforts are taken, almost *any* mode of operation based on an n-bit blockcipher will admit attacks that employ $\sigma = 2^{n/2}$ blocks. Privacy-only encryption schemes achieving beyond-birthday-bound security have been investigated by Bellare, Krovetz, and Rogaway [21], Iwata [95], Jaulmes, Joux and Valette [101], and Minematsu [142],

Do birthday-bound attacks on CBC, CFB, and OFB actually matter? They are of relatively little concern when the blockcipher has a blocksize of $n = 128$ bits, but the attacks can be a serious concern when employing a blockcipher of $n = 64$ bits, requiring relatively frequent rekeying to keep $\sigma \ll 2^{32}$.

4.6. Attacks on CBC, CFB, and OFB when the IV is a nonce. All three modes we are considering in the chapter are trivial to break if the IV is only a nonce. Focusing on the IND$ definition, since that is the only one we have given in full, here are attacks on each of the three modes:

4.6.1 *To break nonce-based CBC.* Consult Figure 4.2. The adversary asks its oracle to encrypt (N^1, P^1) and (N^2, P^2) where $N^1 = P^1 = 0^n$ and $N^2 = P^2 = 1^n$. The adversary outputs one (it thinks it has a genuine encryption oracle) if $C^1 = C^2$. Otherwise it outputs zero.

4.6.2 *To break nonce-based CFB.* Consult Figure 4.3. Assume $s = n$. The adversary asks its oracle to encrypt (N^1, P^1) where $N^1 = 0^n$ and $P^1 = 0^{2n}$, getting back the two-block ciphertext $C^1 = C^1_1C^1_2$. Next the adversary asks its oracle to encrypt (N^2, P^2) where

$N^2 = C_1^1$ and $P^2 = 0^n$. It gets the one-block response C^2. The adversary outputs one (it thinks it has a genuine encryption oracle) if $C^2 = C_2^1$. Otherwise it outputs zero.

4.6.3 *To break nonce-based OFB.* Consult Figure 4.4. The attack just given on CFB also works, but let us instead give a slight variation. The adversary asks its oracle to encrypt (N^1, P^1) where $N^1 = 0^n$ and $P^1 = 0^{3n}$, getting back the three-block ciphertext $C^1 = C_1^1 C_2^1 C_3^1$. Next the adversary asks its oracle to encrypt (N^2, P^2) where $N^2 = C_2^1$ and $P^2 = 0^n$. It gets the one-block response C^2. The adversary outputs one (it thinks it has a genuine encryption oracle) if $C^2 = C_3^1$. Otherwise it outputs zero.

The NIST Recommendation does recognize that a nonce IV is not sufficient for CBC and CFB modes, indicating, at least twice, that the IV must be unpredictable [61, Section 5.3 and Appendix C]. For OFB the Recommendation seems to permit an arbitrary nonce. We hear that "The OFB mode requires that the IV is a nonce, i.e., the IV must be unique for each execution of the mode under the given key" [61, Section 6.4]; and "the IV need not be unpredictable, but it must be a nonce that is unique to each execution of the encryption operation" [61, Appendix C]. Yet we also hear that "IVs for the OFB mode should not be generated by invoking the block cipher on another IV," which begins to sound like we are straying from the idea that any nonce is alright. Note, however, that this language is still not strong enough to banish attack 4.6.3; we did not generate one IV by invoking the block cipher on another IV.

NIST provides some advice on the generation of IVs [61, Appendix B]. Given the frequency with which "wrong" things are used as IVs in CBC mode, this was a good thing to do. In language whose context makes it apparently targeted just to CBC and CFB, the NIST document says:

> There are two recommended methods for generating unpredictable IVs. The first method is to apply the forward cipher function, under *the same* key that is used for the encryption of the plaintext, to a nonce. The nonce must be a data block that is unique to each execution of the encryption operation. For example, the nonce may be a counter, as described in Appendix B, or a message number. The second method is to generate a random data block using a FIPS-approved random number generator. [61, Appendix C, emphasis added]

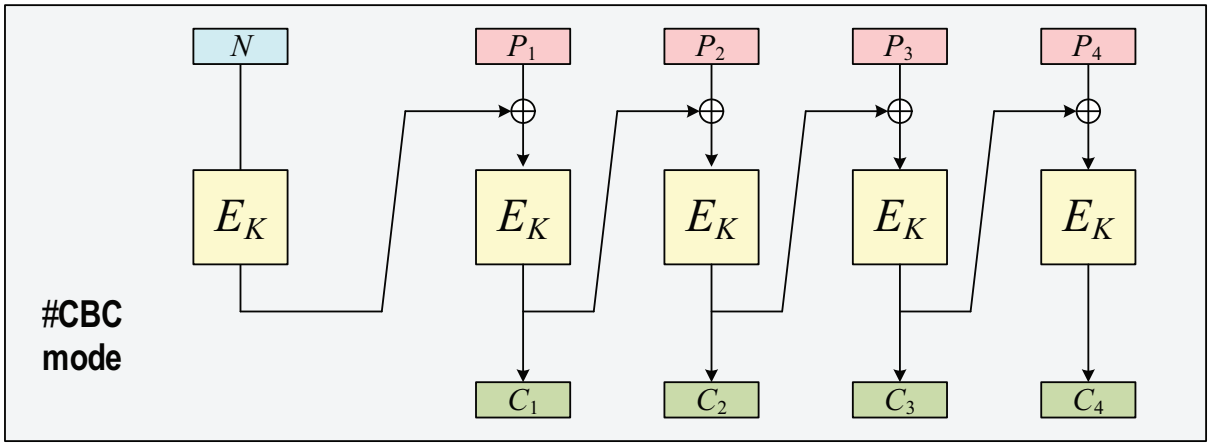
We focus on the first recommendation: to apply E_K to the nonce to make the IV, where K is the key underlying the mode. In effect, this is a *new* mode—a way to turn CBC and CFB (and possibly OFB) into nonce-based schemes ♯CBC and ♯CFB (and possibly ♯OFB). See Figures 4.5 and 4.6.

The modes are not correct as nonce-based schemes. Attacks on ♯CBC and ♯CFB no more complicated than the corresponding attacks on CBC and CFB:

3.5.4 *To break nonce-based* ♯CBC. Consult Figure 4.5. Let the adversary ask a query (N^1, P^1) where $N^1 = 0^n$ and $P^1 = 0^{2n}$, getting back the two-block ciphertext ciphertext $C^1 = C_1^1 C_2^1$. Next the adversary asks a query (N^2, P^2) where $N^2 = C_1^1$ and $P^2 = 0^n$. The adversary outputs one (it thinks it has a genuine encryption oracle) if $C^2 = C_2^1$. Otherwise it outputs zero.

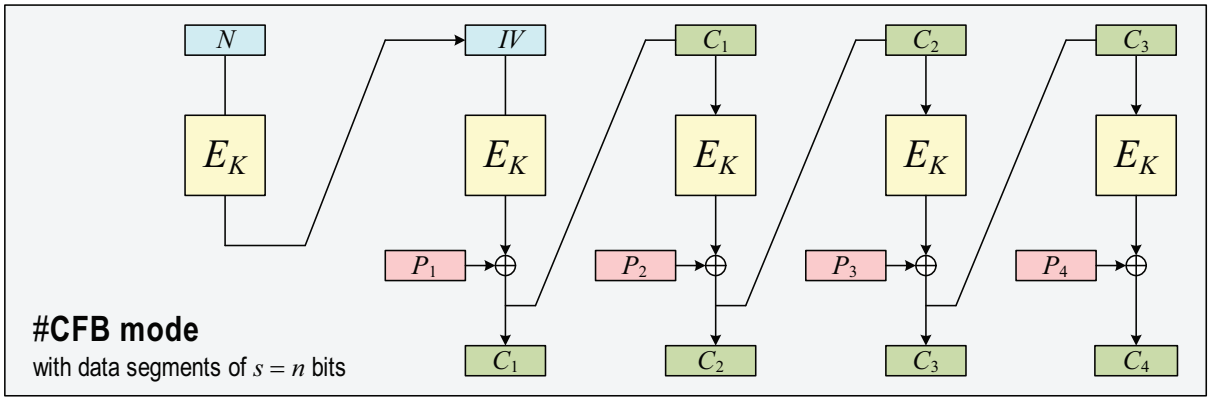
3.5.5 *To break nonce-based* ♯CFB. Consult Figure 4.6. Assume $s = n$. The attack just describes works to break this mode of operation as well.

An explanation for the insecurity of ♯CBC is given by Rogaway [180] (although I did not at that time realize that the mode had been suggested in an Appendix of SP 800-38A). The alternative suggested in that paper is simple: use a different key to map the nonce to an IV, setting $IV = E_{K'}(N)$ where $K \parallel K'$ becomes the key of the mode of operation. In other words, one just replaces the phrase *the same*, in the quoted material from the standard, by *a different*.



40 **algorithm** $\sharp\mathrm{CBC}_K^N(P)$ *♯CBC mode*
41 // $K \in \mathcal{K}$, $N \in \{0,1\}^n$, $P \in (\{0,1\}^n)^+$
42 $IV \leftarrow E_K(N)$
43 $C \leftarrow \mathrm{CBC}_K^{IV}(P)$
44 **return** C

Figure 4.5: **♯CBC mode.** This NIST-suggested variant of CBC is intended for nonce-based encryption. But the mechanism does not achieve the nonce-based notion of security that we have defined.



50 **algorithm** $\sharp\mathrm{CFB}_K^N(P)$ *♯CFB mode*
51 // $K \in \mathcal{K}$, $N \in \{0,1\}^n$, $P \in (\{0,1\}^s)^+$
52 $IV \leftarrow E_K(N)$
53 $C \leftarrow \mathrm{CFB}_K^{IV}(P)$
54 **return** C

Figure 4.6: **♯CFB mode.** This NIST-suggested variant of CFB is intended for nonce-based encryption. But the mechanism does not achieve the nonce-based notion of security that we have defined. The illustration is for the case where data segments have $s = n$ bits.

The attacks on nonce-based CFB and OFB, and on ♯CBC and ♯CFB, can be criticized as requiring IVs that depend on prior ciphertexts. One might claim that no "real" IV—something generated in the encryption process—would have this structure. One could find an intermediate notion, something between nonce-based and probabilistic-encryption, where the IVs are guaranteed distinct and are generated by a process that is independent of the underlying key. If such an exercise is be carried out it seems likely that one could establish a security result for ♯CBC, CFB, and OFB.

4.7. Ubiquitous confusion about IV requirements. As the last section suggests, there are, in SP 800-38A, quite a few questionable assertions about what should be true of modes' IVs. Let me enumerate them. **(a)** For CBC mode, we have a claim that "the integrity of the IV should be protected" [61, Section 6.2]. I cannot figure out what this claim might mean. Who is supposed to do what with the IV? If that suggestion is that one should transmit to a receiver the string $\mathcal{C} = R \parallel C$ in such a way that the R portion is unforgeable, this would seem to be both unreasonable (if you are going to integrity protect the portion R, why not the whole thing?) and unnecessary (nothing in the SemCPA security notion envisions such a requirement). **(b)** It is said that CFB IVs should be unpredictable. While the only provable-security result we have described for CFB is for the case of a random IV, the status of CFB and OFB appear to be the same with respect to relaxing this constraint: a nonce IV is not enough, but a fixed and predictable sequence of IVs, like a counter, looks to be adequate. **(c)** There are suggestions that a nonce IV is enough for OFB. We have explained, this is not the case. **(d)** It is claimed that nonce-based schemes for at least CBC and CFB can be obtained by enciphering the nonce to make the IV, using the same key. We have explained that this is not the case.

Other standards also get these matters wrong. While ISO/IEC 10116:2006—especially its Appendix B—is useful and mostly very well-informed, still it seems full of errors or odd comments on IV requirements for CBC, CFB, and OFB modes. The ISO spec does recommend a "randomly chosen statistically unique" IV for these schemes [92, Sections B.2.1, B.3.1, and B.4.1] (we do not know what "statistically unique" might mean), but it suggests that many other choices for the IV are fine, explicitly including the use of an counter or a message address for CBC or CFB [92, Sections B.2.1 and B.3.1]. This is easily seen to be wrong, and it is quite well-known that it is wrong. Here, for example, is a reference from 1995:

> [It] is not true that CBC encryption can use an arbitrary nonce initialization vector: it is essential that the IV be unpredictable by the adversary. (To see this, suppose the IV is a sequence number: 0, 1, 2, ... Then a (first) encryption of 0x0000000000000000 followed by an encryption of 0x0000000000000001 is recognizably distinct from a (first) encryption of 0x0000000000000000 followed by an encryption of 0x0000000000000000. Clearly this violates the notion of a secure encryption ... [176, p. 8].

Sanctioning a counter IV for CBC is worse than anything in the NIST spec, which never suggests any such thing. Also with respect to CBC, the ISO spec rather elliptically indicates that "[to] prevent information leakage[,] an integrity-protected [and] secret [IV] is recommended" [92, Section B.2.1]. We are aware of no scientific basis for keeping the IV secret and, as before, we cannot guess what integrity protection is supposed to entail, nor what "information leakage" the authors may have in mind.

In fairness to NIST and to the author of SP 800-38A—and also to place things in their appropriate historical context—it is important to emphasize and to appreciate that nonce-based security notions weren't even defined until 2004 [180]—some three years after the publication of NIST SP 800-38A. It should be acknowledged too that numerous cryptography books and papers *also* get wrong what they say about the requirements for the IV in schemes like CBC. Finally, I suspect that I myself did not provide a careful review of NIST SP 800-38A during its period of public review, despite having "lobbied" for the inclusion of CTR mode [122]. The document's author, Morris Dworkin, would no doubt have been receptive to addressing any issues I might have known to point out about the use of nonces in IV-based schemes.

4.8. Malleability of CBC, CFB, and OFB. We do not need to formalize nonmalleability to make it clear just how badly CBC, CFB, and OFB fail with respect to achieving this potential

goal.[1] Recall that, informally, an encryption scheme is malleable if the adversary can modify a ciphertext C for the (possibly unknown) plaintext P to create a ciphertext C' whose plaintext P' is "meaningfully related" to the plaintext P. As an example, with OFB mode, we can take a ciphertext (IV, C) for plaintext M and produce from it the ciphertext $(IV, \overline{C})$ where $\overline{C}$ is the bitwise complement of C. The plaintext for this ciphertext is of course $\overline{P}$, the complement of P. Closely related to P, we have show that OFB is trivially malleable.

It is easy to play such games with CBC, too. Some attacks are trivial; other are somewhat more complex. As an example of the former, note that if $(IV, C_1C_2 \cdots C_m)$ is a ciphertext, each C_i an n-bit block, the message having ciphertext $P_1P_2 \cdots P_m$, then $(IV, C_1 \cdots C_\ell)$ is a ciphertext with plaintext $P_1P_2 \cdots P_\ell$ for any $\ell < m$. We have "mauled" a ciphertext to create related plaintext. Alternatively, runs of ciphertext blocks can be reordered, this resulting in some corruption along the "edges" of each run of blocks but the rest of the corresponding plaintext blocks being similarly rearranged. For a more complicated, and possibly more useful, attack, note that if an adversary possesses a collection of CBC_K-encrypted plaintext/ciphertext pairs then it also has a collection of plaintext/ciphertext blocks $\mathcal{T} = \{(X_i, Y_i) \colon Y_i = E_K(X_i)\}$, the number of these depending on the lengths of the underlying messages. Given $\mathcal{T}$, the adversary can create an infinite number of known plaintext/ciphertext pairs, many of which may be "meaningful." It does this by simply "cutting and pasting" points in $\mathcal{T}$, adding offsets as needed. For example, if $(X_i, Y_i) \in \mathcal{T}$ for all $1 \le i \le q$ then all $(q+1)^m$ sequences $i_0, i_1, \ldots, i_m$ of numbers in $[1, q]$ specify a plaintext/ciphertext pair that the adversary $\mathcal{A}$ can "know"—the one with $IV = X_{i_0}$ and ciphertext $C = Y_{i_1} \cdots Y_{i_m}$, which has plaintext $P = P_1 \cdots P_m$ where $P_i = X_{i_j} \oplus Y_{i_{j-1}}$ for all $j \ge 1$. Some of these plaintext/ciphertext pairs may be "meaningful," and it may be easy for the adversary to find them. See, for example, Stubblebine and Gligor [192] and Bellovin [28] for early work demonstrating the real-world dangers springing from the malleability of CBC encryption.

To show that ciphertexts produced by CFB are malleable, consider, for example, the case where $s = n$. Observe that if (IV, C) is a ciphertext for plaintext P with $|C| = n$, then $(IV, \overline{C})$ is a ciphertext for plaintext $\overline{P}$. Numerous other attacks showing CFB's failure to achieve nonmalleability are just as easy. Note that *no* self-synchronous cipher can achieve nonmalleability; when we "maul" part of a ciphertext, we can be sure that far-removed portions of plaintext are not impacted.

When authors speak of the absence of message *integrity* in a mode like CBC encryption, what they invariably seem to mean is the absence of nonmalleability. True message integrity would coincide with the message authenticity goal: the receiver would know that the ciphertext received (or else the plaintext recovered from it) had to have been sent, in this exact form, by his intended communication partner. Since *none* of the modes CBC, CFB, or OFB are length-increasing and, correspondingly, *all* ciphertexts are valid, achieving message authenticity is out of reach. The best that can be hoped for in this direction is that, when a ciphertext is received, if it *has* been created by an attack, then the plaintext it corresponds to is "junk"—nothing that is predictable or related to other plaintexts received. This is precisely the nonmalleability aim. It is my contention, then, that talk of message integrity, in the contexts of modes like CBC, invariably is talk about a failure to achieve nonmalleability.

4.9. Chosen-ciphertext attacks on CBC, CFB, and OFB. Just as it is easy to break the nonmalleability of CBC, CFB, and OFB modes, it is just as easy to break these schemes in

[1] See Rogaway [180], building on Dolev, Dwork, and Naor [60], for a definition of nonmalleability in the setting of nonce-based symmetric encryption. The chosen-plaintext setting that we here assume is obtained by dropping mention of the decryption oracle.

the CCA (chosen-ciphertext attack) sense. For CBC with a random IV, for example, a simple attack could work as follows. The adversary asks to encrypt a random three-block message $M = M_1M_2M_3$, getting back $(IV, C_1C_2C_3)$. Next the adversary asks to decrypt $(IV', C_1'C_2C_3)$, for random IV', C_1', getting back $M_1'M_2'M_3'$). If $M_3 = M_3'$ then the adversary guesses that it is speaking to a "real" encryption oracle; otherwise it knows that it is not.

It is difficult to overstate the importance of the absence of nonmalleability and CCA security of the IV-based encryption modes; many higher-level protocols go astray from an implicit assumption that the security guarantee from these modes is stronger than it is.

4.10. Padding attacks on CBC and other modes. In an interesting and well-known paper, Vaudenay shows how possession of a "valid-padding oracle" lets one decrypt arbitrary ciphertexts that were encrypted under what one might call the "PAD-then-CBC" mode of operation [194]. The mode of operation encrypts a plaintext P (which need not be a multiple of n bits) by CBC-encrypting the string $\textsc{Pad}(P)$, where $\textsc{Pad}$ is a parameter of the mode. The n-bit blockcipher E remains a parameter as well, and one requires that $\textsc{Pad}(P)$ be a multiple of n bits and that P be recoverable from $\textsc{Pad}(P)$. The envisaged attack model, formalized by Paterson and Watson [164], enriches the usual one by giving the adversary not only an encryption oracle but also an oracle *ValidPadding* that, given $\mathcal{C} = IV \parallel C$, computes P', the CBC-decryption of $\mathcal{C}$ with respect to the underlying key K. If $P' = \textsc{Pad}(P)$ for some plaintext P, then the oracle returns 1, indicating a valid ciphertext—the CBC-encryption of a properly padded plaintext—was provided. Otherwise, the oracle returns 0, indicating an invalid ciphertext.

Vaudenay focuses on the PAD-then-CBC scheme where the $\textsc{Pad}$ function takes in a byte string P and appends to it either `0x01`, `0x0202`, `0x0303030`, and so on, as needed, so that $\textsc{Pad}(P)$ will be a multiple of n bits. For this padding function, Vaudenay shows how, given a *ValidPadding* oracle, one can decrypt an arbitrary ciphertext with a reasonable number of queries. The attack can be seen as a symmetric-setting analog to Bleichenbacher's attack [43] on RSA encryption as realized by PKCS #1.

Black and Urtubia followed-up on Vaudenay's idea and demonstrated that the problem is ubiquitous: they looked at seven additional padding methods, breaking five of the resulting PAD-then-CBC schemes [42]. Canvel, Hiltgen, Vaudenay, and Vuagnoux used Vaudenay's idea to recover passwords in SSL/TLS [49]. Paterson, in a series of works with various coauthors, gives attacks on PAD-then-CBC encryption for ISO padding method #3 [165], for the setting of random and secret IVs [207], and for PAD-then-CBC as realized in IPsec [56, 166]. Mister and Zuccherato [143] and Wen, Wu, and Wen [202] looked at padding-oracle attacks on CFB. (Note, however, that CFB requires no padding if its parameter s is small enough that all messages in the message space will have a multiple of s bits). And one could go on. In general, it is fair to say that, for several years after Vaudenay's paper [194], there was a bit of a cottage-industry investigating how padding attacks on modes of operation could be extended, strengthened, formalized, or made more practical.

Various reactions are possible to this body of work. **(1)** One might assert that there is no problem here: the user was promised SemCPA security and he messed up in expecting something more. Padding-oracle attacks are not outlawed under SemCPA security notions, and one is foolish to think that they won't arise if you start with a malleable scheme. This is the theorist's knowing shrug—correct but not so useful. **(2)** One might assert that one ought, with a mode like CBC, select a padding method for which a *ValidPadding* oracle will be guaranteed useless, perhaps because the oracle will always return an indication of validity [42, 164]. **(3)** Relatedly, one might try to declare, by fiat, that every ciphertext should be regarded as encoding a valid

plaintext, letting decryption return some fixed value when all else fails. In fact, our chosen syntax for an IV-based encryption scheme implicitly demands that something like this be done: we didn't *allow* for decryption to return something other than a point from the message space (see §I.5). But patching decryption in such a way may be infeasible; there may not exist a fixed plaintext that the receiver could pretend to have recovered such that the behavior of a system after recovery of this particular plaintext will not be adversarially recognizable. **(4)** Finally, one might conclude that the problem is ultimately a manifestation of our privacy-only security notion being too weak, as discussed in §I.10. This is the conclusion voiced most often by those who have worked in this domain [42, 56, 146, 165, 166].

4.11. Error-propagation and self-synchronization. In the early days of blockcipher modes, a central operational property of a mode was what would happen if some ciphertext bits get messed up. If a bit gets flipped, we say that there was a *bit error.* A bit error might arise due to transmission across a noisy channel. Or it might occur due to a hardware-failure on a mass storage device. An inconveniently arriving alpha particle could, for example, cause a bit error. Alternatively, some bits might get inserted into or deleted from a ciphertext. Following Alkassar, Geraldy, Pfitzmann, and Sadeghi [2], we call problems of this sort *slips.* As before, a slip might happen during message transmission across a noisy channel. One would usually think of bit errors and slips as being caused by non-adversarial events.

If the occurrence of a bit error will cause only a temporary disruption in the recovery of correct plaintext bits, we say that the mode has limited *error-propagation.* Similarly, if the occurrence of a slip will cause only a temporary disruption in the recovery of correct plaintext bits, we say that the mode is *self-synchronous*, likewise a desirable property in setting where one would be concerned about slips. If one is in a setting where bit errors or slips matter, probably one is hoping for limited error propagation and self-synchronization.

Return to Figure 4.1 for a summary of error propagation and synchronicity properties of CBC, CFB, and OFB modes. The interesting mode is CFB, which not only recovers from bit flips but also slips of s bits where, recall, the parameter s is the data segment size. In particular, a choice of $s = 1$ lets one recover from slips of any number of bits, and a choice of $s = 8$ lets one recover from slips of any number of bytes.

For years I was quite unconvinced that error propagation and self-synchronization were things worth thinking about. In any "modern" design, one might reasonably argue, problems with noisy channels should be corrected *outside* of the cryptography. The architecturally "correct" solution is to first build a reliable channel, perhaps using techniques from the theory of error-detecting or error-correcting codes, and then implement the cryptography on top of the channel. Different concerns, to be handled at different architectural levels by different communities using different techniques. Even thinking about something like self-synchronization felt like a failure to respect an important and desirable abstraction boundary.

While there may be merit to the opinion just expressed, reading Alkassar *et al.* has challenged my intransigence [2]. Those authors explain that there are settings, say in telephony (ISDN is actually the authors' main example), where bit errors or slips do arise, but where there is no need to build a reliable channel, because the occasional bit error or slip simply does not matter. For example, a single bit error or slip might result in an inaudible degradation of sound quality. To deal with such settings in a clean way, a mode like CFB would be attractive—were it not for the inefficiency associated to the scheme.

Here, roughly, is how one might try to understand error-propagation and self-synchronization from a modern perspective. For some parameter $w \in \{1, 2, \ldots\}$, the "locality" of the scheme, one seeks a mode of operation for which, on decryption, each bit $P[i]$ of the plaintext depends only

on the key K and some advancing, width-w "window" into the ciphertext, $C[j]C[j+1]\cdots C[j+w-1]$. CFB with $s = 1$ provides a solution to the problem just named where $w = n$. The "optimized" CFB mode of Alkassar *et al.* provides an alternative solution, one less profligate in its use of the underlying blockcipher [2].

For various reasons, the formulation sketched above is not yet precise. But it is suggestive of a rational approach to studying error-propagation and self-synchronization. It would appear that natural questions in this setting have not been answered, questions like: What is the strongest achievable security notion with locality w? Or: How efficient can we make good schemes with locality w?

I remain skeptical that issues of error propagation or self-synchronization arise in modern designs with any substantial frequency. But I am willing to believe that they arise, and perhaps more often I imagined before.

4.12. Efficiency characteristics. Figure 4.1 enumerates several key efficiency characteristics for CBC, CFB, and OFB. We enumerate some.

4.12.1 While CBC and OFB are "maximal-rate" modes, in the sense that they process n bits of plaintext for every (n-bit blocksize) blockcipher call, CFB processes just s bits per blockcipher call. As a consequence, for $n = 128$, a value of $s = 1$ would mean making 128 times the number of blockcipher calls as CBC, while $s = 8$ would mean making 16 times the number of blockcipher calls as CBC. These is major cost, made all the worse by the awkwardness of paralleling CFB encryption.

4.12.2 Neither CBC nor OFB are parallelizable for encryption, although the first does support parallelizable decryption. CFB does support parallelizable encryption and decryption—any self-synchronizing mode must—but achieving parallelizability is inefficient, involving extra blockcipher calls and latency that depends on the parameter s.

4.12.3 While CFB and OFB decryption can be implemented without ever needing the blockcipher's decryption direction, this is not true for the most popular of the three modes, CBC.

4.12.4 With OFB one can precompute the "pad" that will be xor'ed with the plaintext once it is known. On the other hand, precomputation does not benefit CBC or CFB.

The lack of parallelizability can have a dramatic impact on software performance. In Chapter 5 we will give some data to compare the speed of CTR mode with that of CBC, CFB, and OFB. Figure 5.2 tabulates encryption and decryption speeds for the OpenSSL implementation of the modes running on a recent x86 processor, one supporting fast AES computations. For 1 Kbyte messages,for example, CBC is some 2.8 times slower than CTR mode. Modes CFB (even with $s = 128$) and OFB are slower still.

In short, when efficiency characteristics matter, nothing comes close to CTR: it has better performance characteristics, in multiple dimensions, than any of CBC, CFB, and OFB.

4.13. Concluding remarks. In a 2005 paper by Chris Mitchell [146] provocatively subtitled *Is there a future for CBC mode encryption?*, the author reviews the padding attacks on CBC and the reactions that these attacks have raised. Mitchell maintains that, most of the time, an authenticated-encryption scheme or CTR mode is going to be a better choice than CBC. He writes:

> [O]ur conclusion is that there would appear to be two main choices for the user of a symmetric encryption system: an authenticated-encryption system ... or a stream cipher [eg, CTR].

> This prompts the suggestion in the title of this paper that, *except for legacy applications, naive CBC encryption should never be used*, regardless of which padding method is employed. [146, pp. 255–256, emphasis added].

In short, Mitchell's answer to the question "is there a future for CBC?" is a polite *no*. This would be my answer, too. And the same answer for CFB and OFB. The modes have much legacy value, but relatively little value other than that.

Chapter 5

CTR Mode

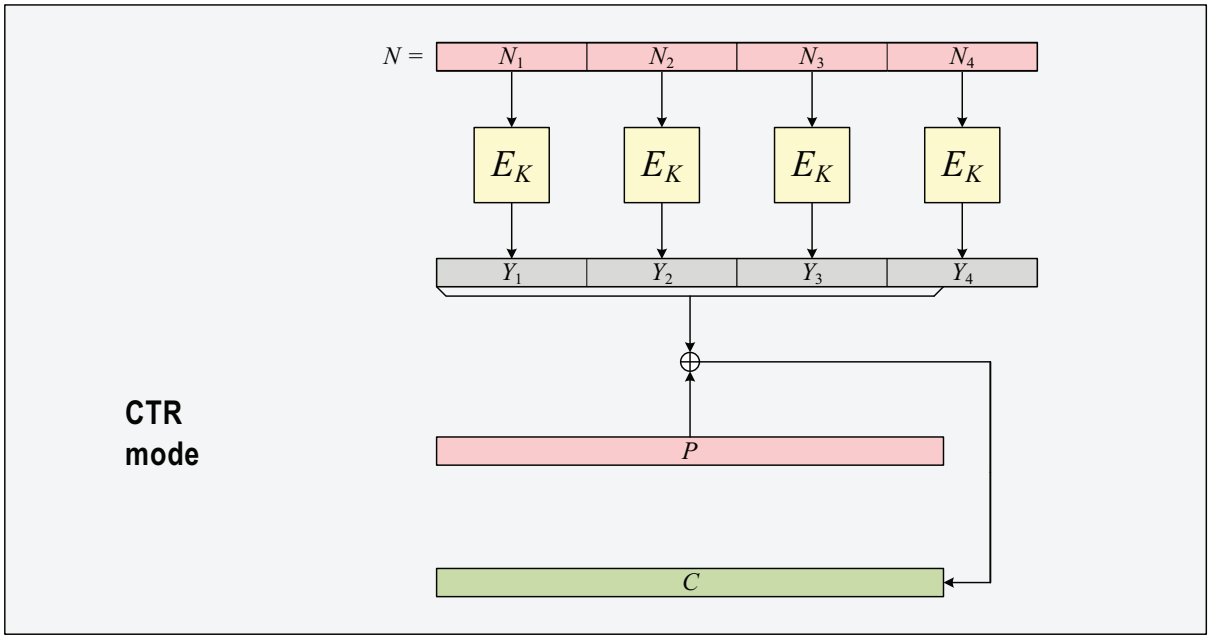
5.1. Summary. This chapter looks at the Counter (CTR) mode of operation, as defined in NIST Recommendation SP 800-38A [61]. The mode is the simplest and most elegant of the confidentiality-only schemes. It was suggested by Diffie and Hellman as early as the basic-four modes [58], yet it was not included in the initial batch of FIPS-approved modes [153]. While I cannot give a definitive explanation for this omission, it may have something to do with a (largely antiquated) belief that a confidentiality mode should provide some sort of nonmalleability guarantee—or else a (completely antiquated) belief that a mode should work to deny adversaries access to chosen plaintext/ciphertext pairs for the underlying blockcipher. NIST added CTR mode to the stock of "Recommended" schemes in December of 2001 [61].

I regard CTR as easily the "best" choice among the set of the confidentiality modes (meaning the set of schemes aiming *only* for message privacy, as classically understood). It has unsurpassed performance characteristics and provable-security guarantees that are at least as good as any of the "basic four" modes with respect to classical notions of privacy. The simplicity, efficiency, and obvious correctness of CTR make it a mandatory member in any modern portfolio of SemCPA-secure schemes.

The only substantial criticisms of CTR center on its ease of misuse. First, it is imperative that the counter-values that are enciphered are *never* reused. What is more, these values are "exposed" to the user of CTR, offering ample opportunity to disregard the instructions. Second, the mode offers absolutely no authenticity, nonmalleability, or chosen-ciphertext-attack (CCA) security. Users of a symmetric scheme who implicitly assume such properties of their confidentiality-providing mode are, with CTR, almost certain to get caught in their error.

In 2000, at a workshop sponsored by NIST, I presented a note written with Helger Lipmaa and David Wagner [122] arguing for the inclusion of CTR mode in the anticipated freshening of standard FIPS Publication 81 [153] (1980). Some of the comments made in this chapter are adapted from that note.

5.2. Definition of CTR mode. Fix a blockcipher $E\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$. This is the only parameter on which CTR mode depends. Our formalization of the mode is then given in Figure 5.1. Regarding CTR as an IV-based encryption scheme (§I.5), we need, as usual, to describe only the encryption direction for the mode; decryption is whatever it "has to be" to be the inverse. CTR mode takes in a key K, a nonce N, and a plaintext P. It outputs a ciphertext C. The message space is $\mathcal{X} = \{0,1\}^*$, the key space is the key space as that for the underlying blockcipher, and the IV space is $\mathcal{IV} = (\{0,1\}^n)^+$, sequences of n-bit strings. The



CTR mode

10 **algorithm** $\mathrm{CTR}_K^N(P)$
11 // $K \in \mathcal{K}$, $N \in (\{0,1\}^n)^+$, $P \in \{0,1\}^*$, $|N| = \lceil |P|/n \rceil$
12 $m \leftarrow \lceil |P|/n \rceil$
13 $(N_1, \ldots, N_m) \leftarrow N$
14 **for** $i \leftarrow 1$ **to** m **do** $Y_i \leftarrow E_K(N_i)$
15 $Y \leftarrow \mathrm{MSB}_{|P|}(Y_1 \cdots Y_m)$
16 $C \leftarrow P \oplus Y$
17 **return** C

Figure 5.1: **CTR mode.** The mode is parameterized by a blockcipher $E\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$. A vector of nonces N must be provided to encrypt a plaintext, the vector having as many components has as the plaintext is long (measured in n-bit blocks). The confidentiality property of CTR will require of any attacking adversary $\mathcal{A}$ that all of the blocks of all of its N-queries be distinct.

number of components of N must be $\lceil |P|/n \rceil$.

It is worth remarking that the decryption algorithm for CTR is precisely the encryption algorithm for CTR, a rather pleasant property. It is also important that the decryption direction of the underlying blockcipher E is never used and that, in fact, the mode makes perfect sense if E is only a PRF, meaning that we drop the requirement of $E_K(\cdot)$ being a permutation.

According to the NIST specification [61], the blockcipher E must be NIST approved, which means, at present, that it must be AES (with any of its three permitted key lengths), Skipjack, or TDEA. The blocksize n used by E will, accordingly, be either $n = 64$ or $n = 128$.

The NIST spec further demands that, as one encrypts or decrypts strings, all of the N_i value—the components of N—must be distinct (across all invocations with the underlying key). For us, this requirement shows up in the security assertion about CTR rather than in the definition of the scheme.

5.3. About the chosen syntax. Rather precisely mirroring the SP 800-38A specification [61], what is input for CTR-mode encryption is not, for example, a single n-bit string, used to indicate where to begin "counting" with this message, but, instead, a whole vector of nonces. This is not the only possible formalization; one might, instead, have conceptualized what is in

the NIST spec by saying that, implicitly, CTR mode has some "hidden" parameter, a *counter-generation algorithm*, Count. The user is free to choose it. The Count algorithm—possibly probabilistic or stateful—would produce the sequence of N_i-values that the encryption scheme will need. The CTR mode would be completely specified once a user specifies both E and Count.

While the "implicit-Count-perspective" may well match what actually *happens* in a system that employs CTR-mode encryption, it is a bit of a reach to say that the spec implicitly demands the existence of such a routine. Our own approach is we think, simpler, stronger, and closer in spirit to the spec. It is simpler because one is never compelled to speak of how counter-values are to be determined; it is simply "not our job." The user will be doing this. It is stronger insofar as we will build into our security notion that *any* counter-generation mechanism is fine—even one that is adversarially influenced—as long as it never creates "colliding" nonce components. And it is truer to the spec insofar as the spec does speak of encryption as needing a sequence of counters (denoted $T_1, T_2, \ldots$ in the spec) but is silent, apart from a non-binding suggestion [61, Appendix B], about how these values might arise.

The decision to not have any sort of built-in counter-generation mechanism means that our formalization of CTR mode is quite different from that of Bellare, Desai, Jokipii, and Rogaway [14]. Those authors assume that the first counter (nonce) (N_1) will be 0, encoded as an n-bit string, and that each successive nonce would be the prior nonce plus one. When the next message comes along, the counter values that will be used for it continue from where the first message left off. Here "plus one" means ordinary integer addition by one, with all integers encoded as n-bit strings. If encrypting a message would entail counting beyond the 2^n possible counter values, the mechanism will refuse to encrypt the message. Thus the BDJR-description of CTR mode was prescriptive, or at least concrete and illustrative, on how counter-values were to arise. This is fine, but it is at odds with the NIST Recommendation, which is intentionally *not* prescriptive about how counter-values are to be chosen.

Despite having to re-formalize the syntax and security property for CTR mode, we like NIST's generality in this setting, which in fact follows our earlier recommendation [122, pp. 1–2].

5.4. Provable security of CTR mode. We already defined our security notion for a nonce-based scheme: indistinguishability from random bits (§I.7), denoted IND\$. We also explained the strength of that notion. We remind the reader that, in the context of CTR mode, a nonce-respecting adversary not only does not repeat nonces—now vectors of n-bit strings—it never repeats any (n-bit) *component* of nonces.

It is simple to lift the result of Bellare, Desai, Jokipii, and Rogaway [14] to our modified syntax and our notion of indistinguishability-from-random-bits. It is an easy exercise to prove the desired result from scratch. Let $\mathcal{A}$ be a nonce-respecting adversary that attacks CTR[E], for some n-bit blockcipher E. Suppose adversary $\mathcal{A}$ achieves advantage $\epsilon = \mathbf{Adv}^{\text{IND\$}}_{\text{CTR}[E]}(\mathcal{A})$ while expending time t and asking queries that total σ blocks. Here "time" should be understood, as usual, to encompass the description size of $\mathcal{A}$, as well as the adversary's running time, while the total number of blocks σ means $\sigma = \sum_{i=1}^{q} \lceil |P^i|/n \rceil$ when the adversary asks queries $(N^1, P^1), \ldots, (N^q, P^q)$. Then there is an adversary $\mathcal{B}$, with a simple and fixed specification in terms of the adversary $\mathcal{A}$, that attacks the underlying blockcipher E and that achieves advantage

$$\mathbf{Adv}^{\text{prp}}_{E}(\mathcal{B}) \quad \geq \quad \epsilon - \sigma^2/2^{n+1} \tag{5.1}$$

where $\mathcal{B}$ asks σ queries and runs in time insignificantly exceeding t (namely, $t + \lambda nq$ for some absolute constant λ depending only on details of the model of computation).

The proof of the result above is based on the observation that CTR mode would be "perfect" (zero adversarial advantage) with respect to the IND$ definition when it is based on a random function instead of a random permutation. This is then combined with an application of the PRP/PRF "switching lemma" (already mentioned in §2.2) and the usual passage between the information and complexity-theoretic settings.

The result above is not only simple; it is also strong, in multiple dimensions.

5.4.1 As we have already explained, we are employing a confidentiality notion that is extremely strong (for an SemCPA notion). Indeed some would argue that indistinguishability from random bits is pointlessly or inappropriately strong—that it attends to something that is fundamentally irrelevant to privacy when it asks for ciphertexts to look random. This may be true, but when going about the business of giving positive results it is perfectly fine to be using an "overly strong" definition. Indeed it is a good thing to do if that notion is actually simpler to work with than more "precise" weakenings, which is the setting we are in.

5.4.2 The provable-security of CTR mode is strong in the sense that the $\sigma^2/2^{n+1}$ loss is tight: there are trivial attacks that do this well. In is strong insofar as we see the exact same sort of quadratic security degradation in every *other* provably-secure confidentiality mode within this report. Not that we are suggesting that one cannot come up with blockcipher-based confidentiality modes that do better than this, as suggested by the following point.

5.4.3 Finally, the provable-security of CTR mode is strong in the sense that the $\sigma^2/2^{n+1}$ security degradation can be regarded as an artifact of using the $\mathbf{Adv}^{\mathrm{prp}}_E$-measure of blockcipher security instead of the $\mathbf{Adv}^{\mathrm{prf}}_E$-measure; if if we switch to the latter, the term goes away. Said differently, we pay the birthday-bound security loss "just" because we started with a blockcipher, rather than a good PRF. While the NIST standard does not allow it, one could, in principle, select a map for E that had excellent PRF-security (it wouldn't be blockcipher) rather than excellent PRP security. Investigations of this idea include [1, 21, 84, 95, 129]. The provable-security of CTR mode provides a simple means of going beyond the birthday bound security loss by processing a blockcipher $E\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ into a PRF $F\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ that intentionally destroys E_K's permutivity and lets F have better PRF-security than would E. Then, of course, one uses CTR[F] instead of CTR[E].

5.5. Desirable features of CTR mode. CTR mode encryption has a number of desirable features. We enumerate some of them below, building on our earlier critique of CTR [122].

5.5.1 **Software efficiency.** Modern processors support some or all of the following architectural features: aggressive pipelining, multiple instruction dispatch per clock cycle, a large number of registers, and SIMD instructions. By eliminating the need to compute C_i prior to C_{i+1} when one encrypts, CTR-mode encryption can achieve major efficiency improvements compared, say, to CBC. In a bitsliced implementation of AES, a mode like CBC is simply impractical; only CTR mode offers up enough parallelism for the technique to be productive [108]. Finally, the "locality" of counters one typically sees with CTR mode—that high-order bits of successive input blocks are usually unchanged (we are assuming a "conventional" increment function) can be used to improve software speed, an observation first exploited by Hongjun Wu [30]. Given all of the above, it is not surprising that, for many blockciphers, a well-optimized implementation of CTR-mode encryption will be considerably faster than a well-optimized implementation of,

	+16	−16	+64	−64	+256	−256	+1K	−1K	+8K	−8K
CBC	4.38	4.43	4.15	1.43	4.07	1.32	4.07	1.29	4.06	1.28
CFB	5.73	5.85	5.56	5.62	5.48	5.56	5.47	5.55	5.47	5.5
OFB	5.42	5.42	4.64	4.64	4.44	4.44	4.39	4.39	4.38	4.38
CTR	5.42	5.42	1.92	1.92	1.44	1.44	1.28	1.28	1.26	1.26

Figure 5.2: **The speed advantages of CTR mode on an x86 "Westmere" processor.** The data is reported on the OpenSSL implementations [162] of the SP 800-38A modes. Positive labels are used for encryption speeds (numbers are byte lengths), negative labels for decryption speeds. The parameter s for CFB mode is $s=128$; smaller values of s will dramatically increase the time.

for example, CBC. Lipmaa finds CTR mode sometimes four times as fast as CBC [121]. This is greater than the gain obtained from switching from the slowest to the fastest AES finalist on most platforms [7].

As a concrete and recent data point, consider OpenSSL's performance for CTR mode encryption compared against CBC [162]. See Figure 5.2 for encryption and decryption speeds using the best-reported speeds on a "Westmere" processor, a machine supporting Intel's AES New Instruction (NI) [77]. For 1 KB messages, CTR mode encryption runs at 1.28 cpb, while CBC encryption runs at 4.07 cpb — about 3.2 times slower. The difference is almost entirely due to the inherently serial nature of CBC compared to CTR, as can be inferred by looking at CBC's decrypt time for messages of the same length, which are essentially the same as CTR's encryption and decryption speed. We note that on on Intel's recently shipped "Sandy Bridge" architecture, CBC performance is going to get still *worse* relative to CTR, since, even though AES instruction may be issued at twice the rate as they could before, their latency will actually go up by 25%, and latency is what kills CBC and is almost irrelevant to CTR.

5.5.2 **Hardware efficiency.** Modes such as CBC encryption are limited in their hardware speed by the latency of the underlying blockcipher. This is because one must complete the computation of ciphertext C_i before one can begin to compute C_{i+1}. Thus the maximal throughput, in hardware, will be about the reciprocal of the latency for the underlying blockcipher E. In contrast, CTR model is fully parallelizable; one can be computing blocks $C_1, C_2, C_3, \ldots$ all at the same time, limited only by the amount of hardware that one throws at the problem. This can result in enormous speedups; in hardware, it would not be at unusual to see CTR mode encrypting at more than 10 times the speed of CBC.

5.5.3 **Preprocessing.** Because the cryptographic work in enciphering a message P is independent of P, preprocessing can be used, in some environments, to increase speed. That is, one can compute the "pad" Y in "spare cycles" even before one knows the plaintext P. When P becomes known, it need only be xor'ed with the already-computed pad Y.

5.5.4 **Random-access**. The ith ciphertext block, C_i, can be encrypted in a random-access fashion. This is important in applications like hard-disk encryption, where bulk data needs to be encrypted quickly. When using CBC mode, properly encrypting the ith block requires one to first encrypt the $i-1$ prior blocks.

5.5.5 **Provable security.** The above efficiency characteristics are not obtained at the expense of security. As explained in §5.4, the "standard" cryptographic assumption about

a blockcipher's security—that it is a pseudorandom permutation—is enough to prove the security of CTR-mode encryption. This is a simple and by now completely standard result [14]. The concrete security bounds one gets for CTR-mode encryption, using a blockcipher, are no worse than what one gets for CBC encryption. Indeed there are approaches to get better security bounds with CTR-mode encryption than with CBC mode, as we have explained, although these do not involve *directly* using the blockcipher E. The security of CTR mode is well-analyzed and well-understood.

5.5.6 **No deciphering.** With CTR mode, both encryption and decryption depend only on E—neither depends on the inverse map $D = E^{-1}$, so D need not even be implemented. This matters most when the inverse direction of the blockcipher is substantially different from the forward direction, as it is for AES. Not having to implement D in hardware can save considerable hardware costs. Not having to implement D in software can reduce the key scheduling time and the context size.

5.5.7 **Arbitrary-length messages**. Unlike other common modes of operation, handling messages of arbitrary bit length is trivial. No bits are wasted in doing this—there is no padding, and the ciphertext C is of the same length as the plaintext M.

5.5.8 **Encryption = decryption.** Encrypting and decrypting under CTR mode are identical, potentially simplifying the amount of code, its runtime size, and the size of hardware embodiments on both FPGAs and ASICs.

5.5.9 **Simplicity**. While inherently subjective, CTR seems, to me, the simplest of the confidentiality modes of operation.

5.6. Objections to CTR mode. The following concerns are sometimes raised with respect to CTR mode encryption. While each objection is valid to *some* degree, I also counter each objection, minimizing the complaint to a significant extent.

5.6.1 **Malleability; no message integrity.** While we have not here formalized any notion for message authenticity, integrity, or non-malleability, it is clear that CTR-mode encryption offers absolutely no such guarantees. Ciphertexts are completely malleable, in the sense of Dolev, Dwork, and Naor [60]. In particular, given the ciphertext C for some plaintext P, an adversary can produce a ciphertext C' for an arbitrary plaintext P' simply by xoring C with $\Delta = P \oplus P'$. This attack is obvious and, sometimes, highly damaging.

That said, it is not normally considered the goal of a confidentiality scheme—certainly none of the schemes in NIST SP 800-38A [61]—to offer up integrity or nonmalleability guarantees. CBC mode, the nominal "alternative" to CTR mode, *also* fails to provide any meaningful message authenticity, integrity, or nonmalleability guarantee. These problems can often be exploited in real-world protocols [192]. If there is any meaningful notion of message authenticity, integrity, or nonmalleability that a scheme like CBC enjoys, it has never been formalized or demonstrated, while attacks demonstrating the absence of any strong notion for message authenticity, integrity, or the malleability are, in contexts like CBC, quite trivial to provide.

When message integrity is desired, the usual approach is to accompany the ciphertext by a MAC (message authentication code), computed under a separate key [23]. When this is done, CTR mode, used in consort with the MAC, achieves the desired authenticity and nonmalleability goals. Alternatively, message integrity or nonmalleability can be guaranteed by using a dedicated authenticated-encryption scheme like

CCM (Chapter 11) or GCM (Chapter 12).

The obvious failure of CTR to achieve message integrity or nonmalleability may in fact contribute to its likelihood of correct use: here more than in CBC mode, it is clear that one is not going to have beyond-confidentiality guarantees.

5.6.2 **Error propagation.** If a bit-flip error occurs in some block of ciphertext, after decryption, the error is localized to the corresponding bit of the corresponding block of plaintext. This is neither good nor bad; it just is. In most applications, it is simply irrelevant. If detection or correction of errors is desired, this should be accomplished at a different architectural level, using appropriate tools, like error-correcting codes. In the modern conception of our field, this simply isn't part of cryptography's role.

5.6.3 **Stateful encryption.** In practice, the vector of nonces to be provided for CTR mode will usually be provided by making the encryption process *stateful*—the party encrypting message will, for example, maintain an n-bit counter.

But the proper use of CBC—or any other IV-based encryption schemes—will typically *also* entail state being maintained, this state further processed to produce the "random" IV. If desired, CTR mode can be implemented with randomness—just like CBC and its cousins. In short, the situation with respect to statefulness in CTR mode is essentially the same as with CBC mode—but the correct use of maintained state is simpler with CTR mode than with CBC.

5.6.4 **Sensitivity to usage errors.** It is crucial in CTR-mode encryption that no component of a nonce-value may be reused. What if a user inappropriately *does* reuse a nonce? Then all security is lost for that second message. But catastrophic errors can arise in *any* mode of operation if the "directions" are ignored. And, it might be pointed out, the nonce-reuse failure for CTR mode is actually *less* catastrophic than it is in a context like CCM (Chapter 11), where reuse of a nonce destroys not only *that* message's privacy, but also all *future* message's authenticity. The recommendations in Appendix B of the NIST document [61] goes some distance to help users understand good ways to construct counters and avoid counter-reuse. And the Recommendation is extremely clear on the perils of reused nonces in CTR-mode encryption.

If one wants a scheme with some genuine robustness to nonce-reuse, one has to go fairly far afield from any of the schemes surveyed in this report. SIV mode, by Rogaway and Shrimpton [183], is a simple scheme designed to provide strong security properties whether or not the provided IV is a nonce (a stronger guarantee is provided if it is).

5.6.5 **Interaction with weak ciphers.** When nonces (counters) are updated in the usual way, as in $N, N+1, N+2, \ldots$, pairs of domain points to which we are applying our blockcipher will usually have small Hamming difference. This may lead to a concern that an attacker can obtain many plaintext pairs with a small and known plaintext difference, which could facilitate differential cryptanalysis [32]. However, this concern is only a valid concern if the underlying cipher is differentially weak. It is not the responsibility of a mode of operation to try to correct—likely without success—for weaknesses in the underlying blockcipher's basic security; that concern should have been addressed when designing the blockcipher.

Another concern we have heard is that since using a counter for the IV in CBC mode is a bad idea (see Chapter 4), maybe CTR mode itself is suspect. This is just sloppy thinking; the problem with using a counter IV in CBC has nothing to do with using counters in CTR mode.

5.6.6 **The standard is too open-ended**. The NIST Recommendation could be criticized for being inadequately prescriptive in *how* the user should construct the sequence of nonces he will use. There is again some legitimacy in this complaint; without indicating just what sequences of nonces should be used, it is more likely that users will provide sequences that do not enjoy the nonce property. Also, interoperability suffers, as does compliance testing and even the providing of good test vectors, when a nonce sequence is not mandated. That said, the advantages to open-endedness are considerable in terms of the versatility of the scheme. When assuming PRP-security for the underlying blockcipher, it is irrelevant what sequence of nonces will be used. Applications of counter mode, including GCM (Chapter 12), exploit the open-endedness of nonce selection in their design.

5.7. Closing comments. In a conversation with Dan Bernstein in 2009, I suggested that it would be useful if the public-domain code associated with his 2008 paper with Peter Schwabe [30] included a similarly optimized implementation for AES decryption; the authors had released their code without implementing the "backwards" direction for AES. Exactly *why?*, Bernstein asked, would someone want to decrypt under AES? Bernstein was suggesting that CTR mode is all that one should ever be doing with AES. While I do not share Bernstein's view—there *are* good uses for the reverse direction of a blockcipher—the basic sentiment that it embodies, that CTR mode is the "right" way to achieve message confidentiality, has a good deal of merit. It is hard to think of any modern, bulk-privacy application scenario where any of the "original four" blockcipher modes—ECB, CBC, CFB, or OFB—make more sense than CTR.

Chapter 6

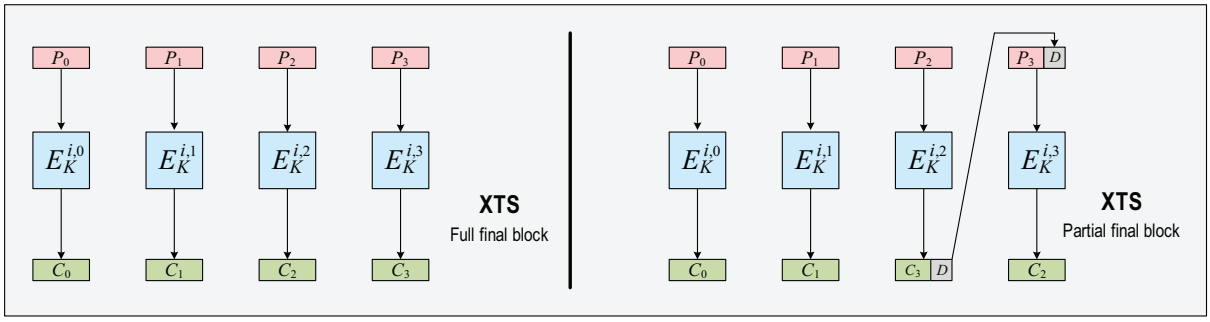
XTS Mode

6.1. Summary. This chapter looks at the XTS mode of operation. The mode is defined in NIST Recommendation SP 800-38E [65] by reference to IEEE Standard 1619-2007 [90]. The latter standard was developed by the IEEE P1619 Security in Storage Working Group, the SISWG. They call XTS a scheme for *narrow-block encryption*. This contrasts to schemes for *wide-block encryption*, which the SISWG is standardizing in a closely related project known as P1619.2 [86].

The intended use of XTS is for encrypting data on a storage device; indeed the mode is not approved for any other use. (We note that XTS is the only mode reviewed in this report for which a particular application domain is mandated.) For this application, some data resident on a mass storage device is divided into a sequence of N-bit strings, for some fixed constant N, the bit length of the *data unit*. For example, the data unit may be a 512-byte disk sector, whence $N = 4096$ bits. Each data unit has a *location*, with locations understood to be given by consecutive 128-bit numbers, beginning with some first location i_0. The data units are encrypted using a key K that is twice the length of the key of the underlying blockcipher, which must be AES-128 or AES-256. The K-encrypted ciphertext for the plaintext P at location i will be $C = \mathrm{XTS}_K^i(P)$. This string will again have $|P| = N$ bits, so that C can replace P on the mass-storage device without disrupting underlying assumptions about how information is located where.

My overall assessment of XTS is mixed. The mode offers significant advantages over "traditional" techniques such as CBC-encrypting each data unit on the disk. That XTS has substantial practical value is attested to by its rapid adoption in at least nine disk-encryption products [205]. In addition, the core idea of partitioning the storage device into fixed-size chunks and applying a tweakable blockcipher, one secure in the sense of a strong PRP, is exactly the right thing to do. Finally, there are significant speed savings for the "narrow block" XTS compared to "wide block" schemes like EME2 [80]: one can expect the former to be about twice as fast as the latter, a difference that might, in some settings, be enough to compensate for the comparatively worse security. All that said, I some have some serious complaints about XTS. The complaints focus on the poor handling of partial final blocks, the small size of the chunks (128 bits) to which we just referred, the paucity of scholarship on the mode, and the absence of well-articulated cryptographic goals for the scheme. I suspect that, for most situations, use of a wide-block encryption scheme, like those soon-to-be standardized by P1619.2, is strongly preferred.

I appreciate and concur with Halevi's sentiment that there is no fundamental difference between what is provided by well-designed narrow-block and wide-block schemes; the difference is quantitative, not qualitative [81]. Yet a quantitative difference in a matter like this may



algorithm $\mathrm{XTS}_K^i(P)$ *XTS mode*
// $K \in \mathcal{K}$, $i \in \{0,1\}^{128}$, $P \in \{0,1\}^N$
$P_0 P_1 \cdots P_m \leftarrow P$ where $m = \lceil |P|/n \rceil - 1$ and $|P_j| = n$ for all $0 \le j < m$, $1 \le |P_m| \le n$
$b \leftarrow |P_m|$
for $j \leftarrow 0$ **to** $m-1$ **do** $C_j \leftarrow \mathrm{XEX2}_K^{i,j}(P_j)$
if $b = n$ **then** *Full final block*
  $C_m \leftarrow \mathrm{XEX2}_K^{i,m}(P_m)$
else *Partial final block*
  $C_m \parallel D \leftarrow C_{m-1}$ where $|C_m| = b$
  $C_{m-1} \leftarrow \mathrm{XEX2}_K^{i,m}(P_m \parallel D)$
endif
return $C_0 \cdots C_m$

algorithm $\mathrm{XEX2}_K^{i,j}(X)$ *Two-key version of XEX*
$K1 \parallel K2 \leftarrow K$ where $|K1| = |K2|$
$L \leftarrow E_{K2}(i)$
$\Delta \leftarrow L \cdot \alpha^j$
return $E_{K1}(X \oplus \Delta) \oplus \Delta$

Figure 6.1: **The XTS mode of operation.** The scheme is parameterized by blockcipher $E\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ and a number $n \le N \le n2^{20}$, where $n = 128$ and E is AES-128 or AES-256. The value i is a 128-bit string that is typically used to encode the sector index.

induce qualitative differences in what security vulnerabilities are not realizable.

When XTS was proposed by NIST, I reviewed it and returned some comments on the mode [181]. After a more thorough review now, for CRYPTREC, I stand by those old comments, and believe that they serve as a good summary of what I will explain, more extensively now, in the current chapter.

> XTS is the two-key version of my XEX construction (Asiacrypt 2004), but operating in ECB mode and with ciphertext stealing for any short final block. While I have no serious objection to NIST approving XTS by reference, I would like to make a couple of points.
>
> First, it is unfortunate that there is nowhere described a cryptographic definition for what security property XTS is supposed to deliver. When the data unit (sector) is a multiple of 128 bits, each 128-bit block within the data unit should be separately enciphered as though by independent, uniformly random permutations. That part is clear. But what security property does one expect for partial final blocks? One might hope, for example, that the final $128 + b$ bits ($b < 128$) would likewise be enciphered as if by a strong PRP. That would seem to be the cleanest natural notion, and it's not too hard to achieve. But XTS does not achieve such an aim (because, for example, C_m does not depend on P_m). One is left to wonder if ciphertext stealing actually works to buy you any strong security property for the final $128 + b$ bits.
>
> Second, I would like to express the opinion that the nominally "correct" solution for (length-

preserving) enciphering of disk sectors and the like is to apply a tweakable, strong PRP (aka wide-blocksize encryption) to the (entire) data unit. That notion is strong, well-studied, easy to understand, and readily achievable. There are now some 15+ proposed schemes in the literature for solving this problem. If NIST approves the "lite" enciphering mode that is XTS, this should not be understood to diminish the utility of standardizing a (wide-blocksize) strong PRP. In the end, because of its much weaker security properties, I expect that XTS is an appropriate mechanism choice only in the case that one simply cannot afford the computation or latency associated to computing a strong PRP. [181]

6.2. Definition of the scheme. The XTS mode of operation is defined and illustrated in Figure 6.1. The construction turns the blockcipher E that operates on n-bit blocks into a tweakable blockcipher XTS that operates on N-bit blocks, where $n \le N \le n2^{20}$. We will define in the the next section just what a tweakable blockcipher is. The blockcipher, E, and the *data-unit's size*, N, are the only two parameters of XTS. The NIST and IEEE specs require that blockcipher E operate on $n = 128$ bits. The blockcipher must be AES with a key length of either 128 or 256 bits.[1]

The XTS cipher takes as input a key K, a tweak i, and the plaintext P, where $|P| = N$ and $|i| = 128$. The length of K is twice the length of a key for E. The constructed cipher returns the ciphertext $C = \mathrm{XTS}^i_K(P)$ where $|C| = |P| = N$. In short, then, the XTS construction with parameter N builds a function

$$\mathrm{XTS}\colon\ \mathcal{K}^2 \times \mathcal{T} \times \{0,1\}^N \to \{0,1\}^N$$

where $\mathcal{T} = \{0,1\}^{128}$ out of a blockcipher

$$E\colon\ \mathcal{K} \times \{0,1\}^{128} \to \{0,1\}^{128}$$

where E is AES with either 128-bit or 256-bit keys. As a tweakable blockcipher, $\mathrm{XTS}^i_K(\cdot)$ is permutation on $\{0,1\}^N$ for any i and K; it can be computed in either the forwards or the backwards direction. Fitting into the syntax of §I.5, the syntax of XTS is that of an IV-based encryption scheme; the IV has just been renamed as a tweak.

In defining XTS we make use of a subsidiary construction, XEX2, defined in lines 30–33 of Figure 6.1. The subsidiary construction is related to a mode called XEX described by Rogaway and derivative of work from Liskov, Rivest, and Wagner [124, 179]. The XTS mode can be regarded as an ECB-like mode over XEX2, but something special—ciphertext stealing [139]—is employed for any fractional final block and its predecessor. We select XEX2 to name the algorithm of lines 30–34, rather than XEX, because the latter construction used one key, not two [179]; the "2" of XEX2 can be understood as a gentle reminder of this difference.

The multiplication operator one sees at line 33, and the implicit (repeated) multiplications in the exponentiation of α on the same line, is multiplication in the finite field $\mathrm{GF}(2^{128})$. The value α is a particular constant in this field, the same constant elsewhere denoted u in this report, or 2 or **2** or $0^{126}10$ elsewhere. For the representation of field points mandated by the standard [90], one can define the needed "repeated doubling" operation for line 33 by

$$L \cdot \alpha^j = \begin{cases} L & \text{if } j = 0, \text{ and} \\ L \ll 1 \oplus [\,135\,\mathrm{MSB}_1(L)\,]_{128} & \text{if } j > 0 \end{cases} \tag{6.1}$$

[1] That AES-192 is not allowed is not explicitly stated in either specification document [65, 90], but the requirement can be inferred from the fact key lengths for XTS must be 256 or 512 bits, and that this is twice the length of the underlying AES key.

where $L \ll 1$ denotes the left shift of the 128-bit string L by one position, namely, $L \ll 1 = \mathrm{LSB}_{127}(L) \,\|\, 0$. The multiplication of 135 and a bit is just ordinary multiplication, generating the result 0 or 135, which is then regarded as a 128-bit string, either 0^{128} or else $0^{120}10000111$. There is actually a byte-swap further involved in the standard's definition of multiplication, done so that little-endian machines will not normally need to do a byte-swap operation. The byte-swap has no security implications, so we ignore it. (My preference, however, would have been to omit the byte swap; other NIST FIPS and Recommendations offer no such favoritism towards little-endian machines.)

NIST indicates that the XTS acronym stands for XEX Tweakable Block Cipher with Ciphertext Stealing [65, p. 2]. The SISWG website recites the acronym a bit differently, as XEX TCB with ciphertext stealing [86]. Here XEX stands for XOR-Encrypt-XOR, and TCB for Tweakable CodeBook mode [86]. The XEX mode of operation is, in turn, due to Rogaway [179], building on work by Liskov, Rivest, and Wagner [124]. As the NIST and IEEE standards are both specific to AES, the full name of the standardized mode is actually XTS-AES (or perhaps XTS-AES128 or XTS-AES256).

6.3. Tweakable blockciphers. Before proceeding with our discussion we should describe *tweakable blockciphers* and their security. The notion was first formally defined and investigated by Liskov, Rivest, and Wagner [124]. The idea had earlier been described and used within a blockcipher construction by Schroeppel [187].

Recall that a conventional blockcipher E has signature $E: \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ and each $E_K(\cdot) = E(K,\cdot)$ is a permutation on $\{0,1\}^n$. In contrast, a *tweakable* blockcipher $\widetilde{E}$ has signature $\widetilde{E}: \mathcal{K} \times \mathcal{T} \times \{0,1\}^n \to \{0,1\}^n$ and each $E_K^T(\cdot) = E(K,T,\cdot)$ is a permutation on $\{0,1\}^n$. The set $\mathcal{T}$, the *tweak space*, is a nonempty set. We let $\widetilde{E}_K^{-1}(T,Y)$ be the unique X such that $\widetilde{E}_K(T,X) = Y$.

The most desirable and useful security notion for a tweakable blockcipher generalizes the strong-PRP security notion for a conventional blockcipher; the adversary must distinguish an oracle for $\widetilde{E}_K(\cdot,\cdot)$ and its inverse from a family of uniformly random permutations, each one named by its tweak, and the corresponding inverse oracle:

$$\mathbf{Adv}^{\pm\mathrm{prp}}_{\widetilde{E}}(\mathcal{A}) \;=\; \Pr[K \xleftarrow{\$} \mathcal{K}:\; \mathcal{A}^{\widetilde{E}_K(\cdot,\cdot),\widetilde{E}_K^{-1}(\cdot,\cdot)}] - \Pr[\pi \xleftarrow{\$} \mathrm{Perm}(\mathcal{T},n):\; \mathcal{A}^{\pi(\cdot,\cdot),\pi^{-1}(\cdot,\cdot)}] \qquad (6.2)$$

where $\mathrm{Perm}(\mathcal{T},n)$ denotes the set of all permutation on $\{0,1\}^n$ indexed by $\mathcal{T}$, so that $\pi(T,\cdot)$ is a uniform random permutation on $\{0,1\}^n$ for each $T \in \mathcal{T}$, all of these permutations independent of one another. Here we re-use the $\pm$prp label in the **Adv** superscript (eg, not bothering to put a tilde over it to distinguish it from the conventional blockcipher setting) because the "type" of argument $\widetilde{E}$ will make it clear whether or not this is the tweakable version of the notion.

The idea captured by the $\pm$prp-definition is as follows: fixing the key, you effectively get a "new" permutation for each tweak, and, despite the common key, all of these permutations behave as though independent. They all look like random permutation, even in the presence of a decryption oracle. Unlike the key, the tweak is not expected to be kept secret.

In applications that use tweakable blockciphers, it is often important that it is fast to change the tweak—faster than changing the blockcipher's key would be. It is desirable if it is especially fast to change the tweaks if the tweaks are being generated in whatever natural sequence arises in the utilizing mode.

6.4. VIL tweakable blockciphers. Tweakable blockciphers as defined in §6.3 to take in a plaintext of some one, fixed length, the blocksize of the blockcipher. There is no problem in

extending the notion to allow inputs of multiple different lengths. (This may sound unnecessary for us to do because n and N are fixed. But be patient.) As a simple example, one might image a tweakable blockcipher that can operate on blocks of either 128 bits or blocks of 192 bits. Syntactically, a tweakable blockcipher will have signature $\widetilde{E}\colon \mathcal{K}\times\mathcal{T}\times\mathcal{X}\to\mathcal{X}$ where $\mathcal{X}\subseteq\{0,1\}^*$ and each $E_K^T(\cdot)=E(K,T,\cdot)$ is a length-preserving permutation. Our $\pm$prp-notion of security can easily be modified to accommodate such an enlarged notion of a tweakable blockcipher: in the reference experiment, one simply defines the idealized object, $\mathrm{Perm}(\mathcal{T},\mathcal{X})$, to include all length-preserving permutations on $\mathcal{X}$, one for each tweak $T\in\mathcal{T}$.

A blockcipher that can operate on strings of multiple lengths is called a *variable-input-length* (VIL) blockcipher. When we speak of VIL-security, as opposed to FIL (fixed-input-length) security, we mean that the blockcipher's operation across these different input lengths do not "interfere" with one another, in the sense that, in the adversary's attack, we allow it to make queries of assorted lengths, whatever is found in domain $\mathcal{X}$. We see no harm in continuing to use the term *blockcipher* in this VIL setting, but we note that sometimes people prefer to avoid the term (saying "cipher" or "enciphering scheme" instead) since, to many cryptographers, the term *blockcipher* is routinely assumed to mean a conventional blockcipher, one that operates on strings of some one fixed length.

We note that XTS itself is not intended to be VIL-secure; its blocklength N is fixed.

See [6, 26, 82] for papers and conceptualizations that run along the lines of this section.

6.5. Three tweakable blockciphers associated to XCB. In the context of XTS there are *three* tweakable blockciphers that we would would like to name, distinguish, and discuss.

6.5.1 The **first** tweakable blockcipher is XTS itself, as defined on lines 10–21 of Figure 6.1. The XTS cipher takes in a key, an N-bit plaintext block P, and a 128-bit tweak i.

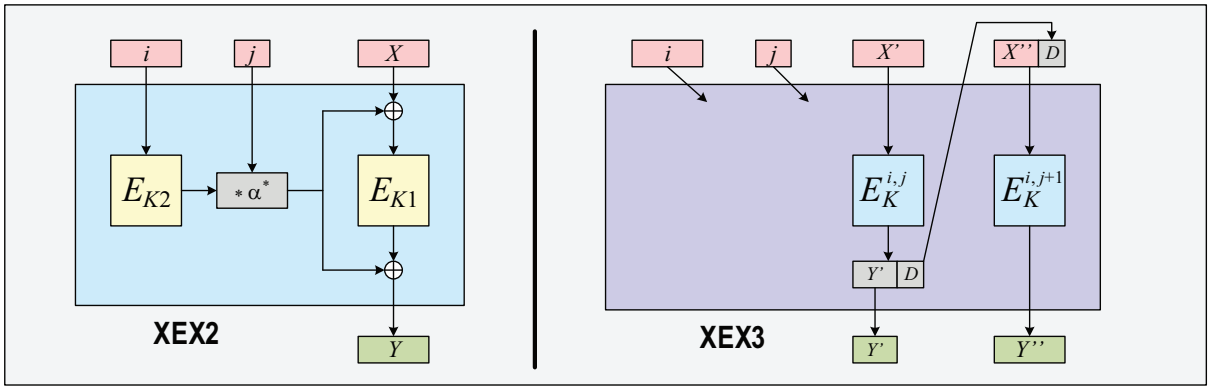
6.5.2 The **second** tweakable blockcipher associated to XTS is XEX2, defined on lines 30–34 of Figure 6.1. The XEX2 cipher takes in a key, an $n=128$ bit plaintext block X, and a tweak $T=(i,j)$ that includes both a 128-bit string i and an integer $j\in[0\,..\,2^{20}-1]$.

6.5.3 The **third** tweakable blockcipher that we associate to XTS is what we call XEX3. See Figure 6.2, where one will find the definition of this VIL blockcipher, and also a description of XTS in terms of it. The XEX3 blockcipher takes in a key, a plaintext block X of 128–255 bits (that is, $|X|\in[n\,..\,2n-1]$), and a tweak $T=(i,j)$ that, once again, includes both a 128-bit string i and a number $j\in[0\,..\,2^{20}-1]$.

The view of XTS associated to its description in Figure 6.2—a scheme that is built on top of XEX3—is, we think, the simplest way to conceptualize what is going on in the mode. In effect, XTS is XEX3 in ECB-mode. Each block gets passed "its" tweak, but all the blocks are treated independently. When XEX3 is called by XTS, only m-indexed blocks can be "long"—they will have $n+b$ bits, where $b=N \bmod n$, while all other blocks will have n bits. Note that in the pseudocode of Figure 6.2 we have changed the indexing so that P_m is a possibly-long final block, as opposed to a possibly-short final block.

One might call XTS, XEX2, and XEX3 the **wide-block**, **narrow-block**, and **narrow*-block** blockciphers associated to the XTS construction. To be clear: while the SISWG calls XTS a narrow-block encryption scheme, the XTS construction itself is *not* a narrow-block scheme; it is a wide-block one.[2] It is, apparently, XEX2 that the SISWG is implicitly referring

[2] The distinction is not a formal one—we never defined "narrow" and "wide"—but, in the context of XTS, one can understand narrow-block schemes as operating on $n=128$ bits, and wide-block schemes as operating on $N>128$ bits.




```
40  algorithm XTS^i_K(P)                                   XTS mode, equivalent description
41  // K ∈ 𝒦,  i ∈ {0,1}^128,  P ∈ {0,1}^N
42  P_0 P_1 ··· P_m ← P where m = ⌊|P|/n⌋ − 1 and |P_j| = n for all 0 ≤ j < m, n ≤ |P_m| ≤ 2n − 1
43  for j ← 0 to m do C_j ← XEX3^{i,j}_K(P_j)
44  return C_0 ··· C_m

50  algorithm XEX3^{i,j}_K(X)                              VIL-tweakable blockcipher underlying XTS
51  // K ∈ 𝒦,  i ∈ {0,1}^128,  0 ≤ j < 2^20,  128 ≤ |X| < 256
52  if |X| = n then                  Ordinary block
53      return XEX2^{i,j}_K(X)
54  else                             Extended block
55      b ← |X| − n
56      X' ‖ X'' ← X where |X'| = n
57      Y' ‖ D ← XEX2^{i,j}_K(X') where |Y''| = b
58      Y'' ← XEX2^{i,j+1}_K(X'' ‖ D)
59      return Y'' ‖ Y'
60  endif
```

Figure 6.2: **Alternative definition of XTS, and definition of XEX3.** The message is partitioned into n-bit blocks and a possibly long blocks. Both are processed by the tweakable blockcipher XEX3.

to when speaking of XTS as a narrow-block encryption scheme. Implicit in thinking of XTS as a narrow-block encryption scheme would seem to be a conflation of the syntax or security properties of XEX2 with the syntax or security properties of XTS itself.

We suspect that the (partly notational) issues that we have tried to straighten out in this section are at the heart of a difficulty to talk carefully about XTS: it is bound to be confusing that there are, implicitly, narrow-block and wide-block tweakable blockciphers on the table at the same time, as well as tweaks of two different forms. And now we have augmented this confusion by describing a third tweakable blockcipher, XEX3, which we regard as implicitly underlying XTS as well.

6.6. Provable security of XEX2. The XEX2 blockcipher on which XTS is built has a good provable-security result. As a reminder, the XEX2 construction (lines 30–34 of Figure 6.1) has tweak space $\mathcal{T} = \{0,1\}^{128} \times [0\,..\,2^{20} - 1]$ and the blockcipher E must be AES. But for thinking about the provable security of XEX2, we will be more general: we will understand the construction as having a tweak space $\mathcal{T} = \{0,1\}^n \times [0\,..\,2^n - 2]$ and think of it as using an arbitrary blockcipher $E\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$.

The security claim about XEX2 is that it is good according to our $\pm$prp-notion. This result follows easily from Liskov, Rivest, and Wagner (LRW) [124, Theorem 2] and the fact that α

is a primitive element of GF(2^n). (Saying that α is primitive means that it generates the entire multiplicative subgroup of this field.) More specifically, consider first the information-theoretic setting, where E_{K1} and E_{K2} are random independent permutations, π' and π. Then the XEX2 construction coincides with the LRW construction [124, Theorem 2] $\widetilde{E}_{\pi',h}(X) = \pi'(X \oplus h(T)) \oplus h(T)$ where the universal hash function employed is

$$h(i,j) \quad = \quad \pi(i) \cdot \alpha^j \tag{6.3}$$

and the multiplication and exponentiation is in GF(2^n), the finite field with 2^n points. To apply the theorem cited we must only assess how good is h as an AXU (almost-xor-universal) hash function, which means asking to bound

$$\epsilon = \max_{\substack{c \in \{0,1\}^n \\ (i_1,j_1) \neq (i_2,j_2)}} \Pr_{\pi} \left[\pi(i_1) \cdot \alpha^{j_1} \oplus \pi(i_2) \cdot \alpha^{j_2} = c \right] . \tag{6.4}$$

Now if $i_1 \neq i_2$ then it is easy to verify that $\epsilon \leq 1/(2^n - 1)$, and if, instead, $i_1 = i_2$ and $j_1 \neq j_2$ then, once again, the same bound holds. As a consequence, [124, Theorem 2] says that, when using a pair of random permutations to underlie XEX2,

$$\mathbf{Adv}^{\pm\mathrm{prp}}_{\mathrm{XEX2}[\mathrm{Perm}(n),\mathrm{Perm}(n)]}(\mathcal{A}) \quad \leq \quad \frac{3q^2}{2^n - 1} \tag{6.5}$$

if the adversary $\mathcal{A}$ makes at most q queries. Passing to the complexity-theoretic analog, where one assumes uses a "real" blockcipher E and adds in a term for twice the ±prp-insecurity, is standard. We comment that it is straightforward to drop the ugly -1 in (6.5) by noting that the proof of [124, Theorem 2] in the paper's appendix already gives up more than that in simplifying the theorem statement.

For results closely related to the strong-PRP security of XEX2, or to improve the constant "3" in (6.5), see Minematsu [141] and Rogaway [179]. But part of the point of the description above is to emphasize that, when using two keys, things are simpler than with XEX, and one need look no further than Liskov, Rivest, and Wagner [124] to find the needed proof.

6.7. Security of XEX2 is not enough. Does it follow from the result just given—the fact that XEX2 is a "good" tweakable blockcipher—that XTS does what it should do (assuming AES itself is good as a strong-PRP)? The answer is *no*, for two related reasons. The first is that we haven't said *what* XTS is supposed to do. It might have nothing to do with XEX2 being "good." The second is that XTS is *not* XEX2; it is a tweaked-ECB-with-ciphertext-stealing construction that is *based* on XEX2. Whatever properties XEX2 might, we'd have still to show that XTS correctly uses XEX2 for whatever *its* aims may be.

6.8. Security goals for XTS. What is it that XTS *should* achieve? There are no easy answers to be found in the IEEE or NIST documents [65, 90], only suggestive hints.

The "syntactic" expectation for XTS is clear and explicit in the IEEE spec [90]: the SISWG sought a tweakable blockcipher, the tweak being the data-unit's location. This makes good sense. But what security property of this object? There are multiple indications that the goal reaches beyond confidentiality. Recounting points from [90, Appendix D.2], we know that, in designing XTS,

6.8.1 CTR mode (implicitly, the counter values depending on the data-unit's location) was dismissed because of its being *trivially malleable*: the ciphertext C having plaintext P can be modified to a ciphertext C' having any desired plaintext P' by xoring C with $\Delta = P \oplus P'$.

6.8.2 CBC mode (implicitly, the IV based on the data-unit's location) was dismissed because a ciphertext stored at one location could be moved to another, and the newly relocated ciphertext will now decrypt to the *same* value as the original ciphertext, apart from the first block. (Recall that, with CBC mode, the plaintext block corresponding to ciphertext block j depends only on the key and ciphertext blocks $j-1$ and j.) This kind of attack should again be considered as a nonmalleability attack [60, 82].

6.8.3 CBC mode (implicitly, the IV based on the data-unit's location) was also dismissed because an adversary can intentionally flip a bit of a plaintext by flipping the corresponding it of the ciphertext. This also side-effects one *other* block of plaintext, but this side effect might not always matter. Once again, this attack can be considered an attack on the nonmalleability of the tweakable blockcipher that one aims to construct.

In consideration of the attacks above, one might think that privacy and nonmalleability are the twin goals of XTS. But such a claim would be too facile, for a number of reasons.

6.8.4 To begin, it's not even clear what privacy means in this setting. We have in XTS a deterministic ECB-like mode of operation, so we *know* that it will leak not only repetitions of (N-bit) plaintext blocks, but even repetitions of particular (n-bit) chunks of them.

6.8.5 Similarly, it is not clear what nonmalleability means in this setting; there are inherent nonmalleability attacks for *any* ECB-like scheme. Concretely, nonmalleability is supposed to capture the idea that an adversary cannot modify a first ciphertext to create a second ciphertext whose underlying plaintext is meaningfully related to the underlying plaintext of the first ciphertext. But, with a scheme like XTS, one most certainly *can* modify a ciphertext in such a manner: for example, flip a bit of the first block of ciphertext and you will, with certainty, change the first block of plaintext, but nothing else. This *is* a nonmalleability attack of the cipher.

6.8.7 Finally, as we will explain in a moment, the privacy and nonmalleability goals are not exhaustive of XTS's security aims.

We can conclude that whatever might be meant by XTS achieving privacy and nonmalleability, the definition for such notions would be thoroughly contingent on the fact that XTS is an ECB-like scheme; privacy and nonmalleability would be understood to mean that the scheme is somehow as private and as nonmalleable as you could hope to get in such a setting. But, in fact, this won't be true; XTS will *not* be as private or as nonmalleable as one might hope to get in an ECB-like mode. The problem has to do with the handling of the partial final blocks—the ciphertext-stealing solution that was employed. The technique is too weak to achieve a strong notion of privacy or nonmalleability, as we will later sketch out.

* * *

We have, so far, only indicated that there are privacy and nonmalleability goals for XTS. But it should also be acknowledged that there are other, still less rigorously explained or understood goals for XTS.

As an example of a potential goal, consider the Wikipedia article "Disk Encryption Theory" [205], which complains that CBC suffers from a "watermarking problem" wherein an adversary can choose to write some data units (disk sectors) whose ciphertexts in some way "spell out" some particular, adversarially-chosen value—the adversary has "watermarked" the disk despite its being encrypted. But this is *always* possible, under the sort of attack model provided by our adversary. One uses the folklore trick for creating a covert channel; for example, write a

```
70  algorithm F_XTS(i, C)                                          Filter for XTS
71  C_1 ··· C_m ← C where |C_j| = n
72  for j ← 1 to m do
73      if T[i, j, C_j] = undefined then T[i, j, C_j] ← Cnt_j^i ,  Cnt_j^i ← Cnt_j^i + 1
74      P_j ← [T[i, j, C_j]]_n
75  P ← P_1 ··· P_m
76  return P
```

Figure 6.3: **Filter function for XTS mode.** The program formalizes that which we *know* to leak by the mode—for each tweak i, repetitions of each message blocks. See Chapter 3 for a fuller description. We assume here that all messages are a multiple of n bits.

sequence of random plaintext sectors whose first bits of ciphertext "spell out" whatever message one wants to send. Use trial-and-error to find such plaintext blocks. As best I can determine, there is no achievable substance to this adversary-can-watermark complaint; this "vulnerability" is implicit as soon as one imagines that an adversary can read ciphertexts sectors and write plaintext sectors.

While the can't-leave-a-watermark goal may seem to be reaching in terms SISWG's goals for XTS—nothing along these lines is mentioned in the spec [90]—it is clear that privacy and nonmalleability aims do not exhaust the Working Group's cryptographic goals for XTS. Most significantly, the XEX2-based method within XTS replaced what had been called the LRW scheme [8, Slide 13] because of what happens in the latter mode if one encrypts part of the key. In particular, with LRW (an instantiation of what is in [124]) one has a tweakable blockcipher $\widetilde{E}^i_{K1\ K2}(X) = E_{K1}(X \oplus \Delta) \oplus \Delta$ where $\Delta = i \cdot K2$ and $i \in \{0,1\}^{128}$ is the tweak (the value i an n-bit string specifying a location on the storage device). The SISWG made the LRW→XEX2 transition after it was recognized that, with LRW, the encryption of key $X = K2$ would cause the scheme to completely fail [81, Slides 18–19]. The irony is that XEX *too* has no provable-security property along these lines; if it enjoys some sort of KDM (key-dependent message) security [40, 48], that would be purely by accident. In fact, such an accident does take place: at least in the ideal-cipher model, it seems possible to prove that XEX and XEX2 *are* KDM-secure.

Summarizing, at this point we have at least three informally-understood cryptographic goals for XTS: (1) (limited) privacy (that which is achievable given for an ECB-like tweakable blockcipher); (2) (limited) non-malleability (again, that which is achievable for an ECB-like tweakable blockcipher); and (3) some sort of KDM-security—at least something strong enough to allow the encryption of ones own key, or parts thereof. Of these three properties, none has been formalized in a way appropriate to the current context, and none has been shown to be achieved by XTS. In short, there is a fairly large gulf between the security of XEX2 as a tweakable blockcipher and the security of XTS.

6.9. The analytic dilemma. What would next happen in a rigorous treatment of XTS or, more generally, the problem that XTS aims to solve? One natural path would run about like this:

- One might begin by formalizing the best-achievable-privacy for a tweakable ECB-like scheme, where one assumes that messages are a multiple of the blocksize n. In Chapter 3 we showed how to formalize this sort of thing for ECB; in our preferred approach, we defined a "filter" $\mathcal{F}$ that effectively captures what we *know* a ciphertext to leak, encoded in the form of a canonical plaintext that is associated to each ciphertext. We then talk about being private up-to that which $\mathcal{F}$ reveals. The approach is easily adapted to our setting

using the filter $\mathcal{F}_{\mathrm{XTS}}$ defined in Figure 6.3.

- One could go on to formalize the best nonmalleability achievable by a tweakable ECB-like scheme, where one assumes that messages are a multiple of the blocksize n. This does not appear overly difficult; one takes a definition for nonmalleability in the tweakable-blockcipher setting [82] and blends this with the filter-function idea, using the same filter $\mathcal{F}_{\mathrm{XTS}}$ associated to privacy.
- One could then work out a notion for KDM-security in the context of an ECB-like tweakable blockcipher. This begins to sound rather esoteric and ungainly; it makes more sense to first develop KDM-security in the context of a PRP-secure (tweakable) blockcipher (KDM-security is a notion normally spoken of with respect to probabilistic encryption schemes, either in the symmetric or asymmetric setting [9, 40, 44, 48]).
- One would show that applying a tweakable blockcipher secure as a strong-PRP works to achieve the now-defined notions. One would expect this to be routine, if good definitions were in place.
- To cover the problem XTS aims to solve, one would now have to generalize all of the above to the case of ciphers that act on messages that are not multiples of n bits. This sounds highly unpleasant, something that would lead to technical and unconvincing definition.

The original conception for this chapter actually envisaged doing all of the work outlined above. But the effort amounts to writing a rather substantial academic paper on XTS and, ultimately, one that would probably not be very interesting. Fundamentally, the source of unpleasantness is that one simply shouldn't have to go here, where the "here" entails focusing on an ECB-like mode-of-operation instead of the mode's central constituent part. We explain it as follows:

> **The analysis-of-XTS dilemma:** XTS is defined, syntactically, as a particular wide-block, AES-based mode of operation, so an analysis of XTS is in some sense *obliged* to focus on this mode. But specifying security properties for this kind of object, properties achievable for an ECB-like mode, is something both awkward and new. It would be cleaner—and more responsive to the commonality between wide-block and narrow-block schemes—to focus just on the narrow-block "part" embedded in XTS. But what *is* this narrow-block part? It doesn't work to think of it as XEX2, since fractional blocks and their predecessors are not directly processed by XEX2. It would seem that XEX3 is the "real" constituent part of XTS. Yet XEX3 does not have strong security properties—in particular, it is not a strong PRP. Thus when one draws the abstraction boundary where it seems that it "should" be drawn, all one can say is that the constituent part does not do what it ought to do. This is not a very satisfying answer.

In the next section we explain what XEX3 "ought" to have achieved, and why.

6.10. What the embedded narrow*-block scheme "should" have achieved. In order for one to say something simple and meaningful about XTS, it would be desirable to make the proclamation that all of the separately-encrypted "chunks" on the mass-storage device are enciphered by a strong, tweakable-PRP. Perhaps this was the SISWG's idea, but they didn't attend well to fractional final blocks. What we would like to say is that XEX3 is a strong-PRP on its domain, where its domain has strings of two lengths: n bits ($n = 128$) and $n + b$ bits (where $b = n + (N \bmod n)$). Things would be pretty simple and clear if this claim were true. In particular, one could go on to establish that a tweakable strong-PRP achieves good privacy and non-malleability properties, if one wanted to do so. For a well-designed scheme, one could even

establish KDM-security, in the ideal-cipher model. We note, in particular, that it seems true—and straightforward to prove—that XEX2 achieves KDM-security in the ideal-cipher model. Thus, by postulating a simple abstraction boundary—a (slightly) VIL blockcipher, one that works on either n bits or $n + b$ bits—and by constructing a strong-PRP that fits this syntax, we would be able to say something reasonably simple and strong.

Why is XEX3 *not* a strong-PRP? Of course it *is* a strong-PRP if N is a multiple of n (which may well be the customary case). This was the result discussed in §6.6. But when we are dealing with an extended block, it is trivial to break XEX3 as a strong-PRP, or even an ordinary PRP. Simply note that, given an $n + b$ bit plaintext block, the last bit of ciphertext does not depend on the last bit of plaintext, a property that trivially allows one to distinguish XEX3 from a random permutation on $n + b$ bits. The attack should not be viewed as artifactual, flowing from a too-strong notion of security: if one defines privacy and nonmalleability goals for our $(n + b)$-bit blockcipher, it's going to be possible to break them for XEX3.

Would it have been possible to redesign XEX3 so as to make it was a strong-PRP? Sure; there are multiple approaches in the cryptographic literature, although there is nothing that we would regard as a refined and maximally-efficient scheme. Here are some "canned" approaches for how to encipher messages of either n or $n + b$ bits (where $b \in [0 \,..\, n - 1]$) starting from an n-bit blockcipher.

- Use the Luby-Rackoff construction (a Feistel network with, here, an AES-based round function) with four or more rounds [127, 163].
- Use the XLS construction from Ristenpart and Rogaway, which would employ three blockcipher calls and a simple mixing function based on a pair of "orthogonal Latin squares" [175].
- Use the EME* (aka, the EME2) construction from Halevi for this $(n + b)$-bit string [80]. The mode is poised to be standardized under IEEE P1619.2.
- Use some fully-instantiated version of the NR-mode [150, 152]

Note that it is important in some cases to use separated keys for the n-bit enciphering setting and the $n + b$ bit one. And, for all of these starting points, we are not suggesting that it is trivial to go from what is in the literature to a nice, standardized scheme. For one thing, it should be acknowledged that all of the solutions above are going to come out more complex and less efficient than what XTS does, a simple two-blockcipher-call solution based on ciphertext stealing. For example, the first of the suggestions will need at least four blockcipher calls, rather than two. I suspect that there are relatively simple two-blockcipher-call constructions for enciphering a string of between $n+1$ and $(2n-1)$ bits, achieving strong-PRP security if the underlying n-bit blockcipher does. Indeed I comment on this in a recent paper [185, Footnote 4], but fail to offer up a concrete solution.

Possibly SISWG participants were aware of the possibility of providing a "strong" solution for enciphering the final $n + b$ bits, but either viewed this case either as unworthy of much attention or as unworthy of extra computational work. And I cannot assert with any certainty that such a decision is wrong. All I can say is that what was done leaves one in a bit of a quandary for saying anything simple, strong, and rigorous about XTS.

6.11. Weakness of ciphertext-stealing in the context of XTS. More concretely, let us suppose we will be enciphering data-units of one full block and one fractional blocks using XTS. For clarity, let us omit the swap of the final full and partial blocks of ciphertext (that is, replace

the return of $Y'' \parallel Y'$ by a return of $Y' \parallel Y''$ at line 59 of Figure 6.2), which has no security significance and is explicitly not required in the NIST standard [65].

In the following adversarial attack the data-unit location will be fixed in all adversarial queries, so we ignore it. We assume $N = 255 = 2n - 1$. Let the adversary ask an encryption query of $A \parallel a$ where $|A| = 128$ and $|a| = 127$ for random A, a. The adversary gets back a ciphertext of $b \parallel B$ where $|b| = 127$ and $|B| = 128$. Now let adversary asks a decryption query of $b \parallel B'$ where $|B'| = 128$, for a random B'. The adversary gets back a plaintext $A' \parallel a'$ where $|A'| = 128$ and $|a'| = 127$. We observe that, half the time, $A' = A$.

Is this an attack on privacy? At some level, it is impossible to answer such a question; as we have already belabored, XTS has no formalized security goal. Maybe this is just a "mix-and-match" attack of the sort that one *knows* that any ECB-like scheme will be vulnerable too. But note that we have mixed-and-matched strings with a granularity of less than 128-bits, which doesn't sound like it "ought" to be possible. We're trying to get some sort of minimum-granularity of at least 128 bits.

The above is worded to sound like a chosen-ciphertext attack on privacy. We can play the same sort of games with malleability. The adversary is not supposed to be able to create ciphertexts that have plaintexts that are "meaningfully related" to the plaintexts for other ciphertexts, where "meaningfully related" would somehow have to capture something that is not unavoidable with an ECB-like scheme. But suppose the adversary asks plaintext $A \parallel a$, just as before, getting back ciphertext $b \parallel B$. The adversary creates ciphertext $b \parallel B'$, just as before. Its plaintext is, often, quite closely related to the plaintext for a different ciphertext: half that time, the plaintext begins with A.

This section should perhaps bolster the belief that worrying about notions like privacy and nonmalleability of ECB-like modes is the wrong way to go; it is going to be clearer and more compelling to deal only with strong-PRP security.

6.12. Comments to NIST from others. Liskov and Minematsu prepared rather extensive comments on XTS for NIST [123]. Their main criticism was the choice to using two keys: XTS's selection of XEX2 rather than the original XEX. We agree that this choice is not well justified by the spec [90], but we have never regarded the reduction in the number of blockcipher keys as a clear or important win. The key-reduction tricks complicate proofs and conceptual complexity while they improve an efficiency characteristic—key length—of unclear importance. We comment that recent SISWG meeting minutes (2 Dec 2010) show that the group is working to add-in a one-key version of XTS. We don't know why.

The more important critique from Liskov and Minematsu, as far as we are concerned, is the comment that "There is insufficient analysis of the mode of operation, that is, of the use of sequential-tweak ECB with ciphertext stealing" [123, p. 4]. This is one of our points, but here said quite a bit more compactly. Liskov and Minematsu nonetheless suggest that the ciphertext stealing approach is sound [123, Section 3.4]. My own conclusion is different, but it is bound to depend on the security notion one (implicitly or explicitly) aims for.

The NIST specification document for XTS [65] is the only NIST Recommendation I have ever read that is *not* self-contained: it defines XTS by reference to the IEEE specification document [90]. This would be undesirable but not a big deal if the IEEE specification were simply and freely available from NIST and the IEEE. Instead, the only on-line versions of the IEEE spec that I can locate on the web may well be in violation of the IEEE's copyright. Vijay Bharadwaj, David Clunie, Niels Ferguson, and Rich Schroeppel all comment to NIST on the undesirability of having a NIST standard defined in a way that its only legal acquisition is at

cost [31, 52, 174]. I concur.

Vijay Bharadwaj and Neils Ferguson make a number of insightful comments on XTS [31], the most interesting of which speaks to how the narrowness of the tweakable tweakable blockcipher, just 16-bytes, can give rise to practical attacks. First the authors sketch how to manipulate the code for an encrypted program so that its decryption will, with reasonable-probability, provide a security vulnerability, not just a crash or exception. With a wide-block encryption scheme, this would appear to be considerably more difficult. The authors similarly explain how the narrowness of the tweakable blockcipher lets one more accurately and usefully corrupt encrypted data. Overall, the thrust of their comments is that, even if narrow-block and wide-block schemes are only quantitatively different, the practical impact of ECB's malleability in the two settings is very different.

6.13. Closing remarks. A crucial question we have only just skirted is whether or not it actually makes sense to use a narrow-block scheme, XTS, instead of a wide-block one, like EME2, because the former needs less computational work. The computational savings is real—one can expect EME2 to run at about half the speed of XTS. But the security difference is real, too, and harder to quantify. The question of whether the tradeoff is "worth it" is more a security-architecture question than a cryptographic one, and the answer is bound to depend on the specifics of some product. Yet if one is going to be reading or writing an entire data-unit, spending more work than to ECB-encipher it already, it seems likely to be preferable to "do the thing right" and strong-PRP encipher the entire data unit. Were I consulting for a company making a disk-encryption product, I would probably be steering them away from XTS and towards a wide-block scheme. "Too expensive" is a facile complaint about bulk encryption, but often it's not true.

If the latency issue is actually a problem in some application setting, a well-designed on-line cipher [12, 185] might be only marginally slower than XTS. It would sit logically between a narrow-block scheme and a wide-block one in terms of the security property that is delivered. But it seems likely that an online cipher will still represent an unnecessary security-property compromise for most application settings.

In the end, XTS is probably a useful mode, one that offers significant advantages to previous prevailing practice. But were one to standardize only one mechanism for length-preserving, data-at-rest encryption, a wide-block enciphering scheme would be a better choice. Probably it is just "historical accident" that a narrow-block scheme is available before a wide-block one, making it harder to ignore the former and go with the latter.

Part II

Authenticity Modes

Part II

Authenticity Modes

The following four chapters will evaluate some message authentication codes (MACs), treating, in turn, the CBC-MAC variants of ISO 9797-1, CMAC, HMAC, and GMAC. Here we bring together common preliminaries for this material.

II.1. Syntax of a MAC. For our purposes, a *message authentication code* (MAC) is an algorithm that takes as input two arguments: a key K and a message M. Both are binary strings. The algorithm outputs a *tag* (or, giving the word a second meaning, a *MAC*). The tag is another binary string. It's length is some fixed number of bits, τ, which we call the *tag length*. We can thus write the signature of MAC F as $F\colon \mathcal{K} \times \mathcal{M} \to \{0,1\}^\tau$.

The syntax just described is for a *deterministic* MAC. There are also MACs that are probabilistic, stateful, or nonce-based. They do not concern us in this report except for GMAC, which is a nonce-based MAC. We will defer an explanation of that wrinkle until Chapter 10; here we focus on deterministic MACs.

For some MACs, values like that tag length are regarded as *parameters* of the scheme. Such values are expected to be fixed within some application, so they are not considered to be arguments of the MAC so much as deferred choices for arriving at the (fully-specified) algorithm. Sometimes we may write parameters in brackets, following the name of the MAC. When we speak of a MAC we may, depending on context, either be thinking of the object with the parameters specified or unspecified. The former view is used when we need to have a MAC match the formal syntax we've described; the latter view is used when we need to have a MAC match the description one sees within a specification document.

Besides the tag length, the underlying *blockcipher* is a parameter on which any blockcipher-based MAC will depend. The MACs reviewed for this report are all blockcipher-based except for HMAC, which is based on a cryptographic hash-function instead. In fact, most cryptographic hash functions are also based on blockciphers, so one may, in fact, look upon HMAC as blockcipher-based, if its underlying hash function is. Classically, though, one thinks of HMAC as having a different starting point than CMAC, GMAC, or any of the CBC-MACs.

II.2. Security of a MAC. The standard security notion for a MAC is quite simple: it is security against an adversary that would like to find a single forgery ("existential forgery") after it carries out an adaptive chosen-message attack (ACMA). Let F be a MAC, meaning, again, an algorithm with signature $F\colon \mathcal{K} \times \mathcal{M} \to \{0,1\}^\tau$ that maps a key K and message M to its tag $T = F_K(M)$. To define security, first, a key K is chosen at random from the key-space $\mathcal{K}$. Then an adversary $\mathcal{A}$ is given oracle access to a pair of oracles:

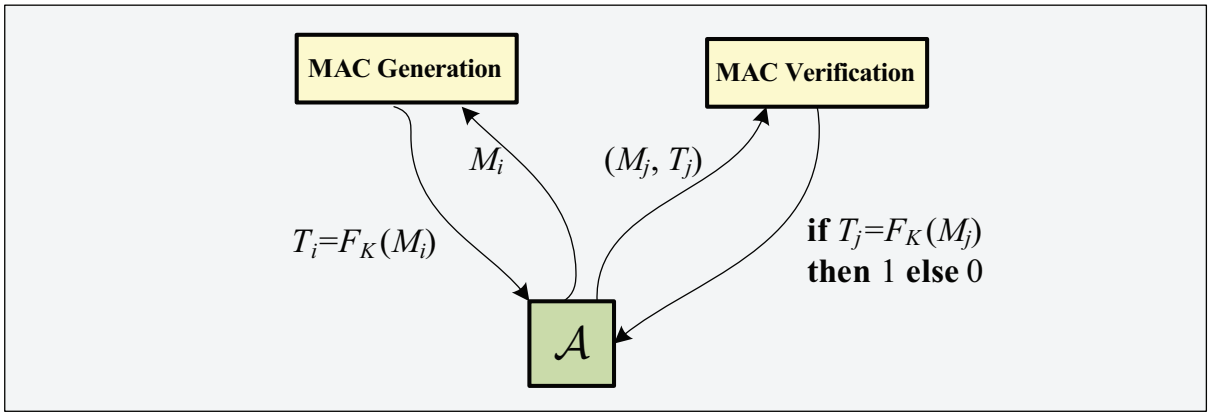


Figure 6.4: **Model for MAC security.** The adversary $\mathcal{A}$ may ask a sequence of MAC-generation queries interleaved with MAC-verification queries. It's goal is to *forge*—to ask a "successful" (1-returning) MAC-verification query (M_j, T_j) for a message M_j never asked to the MAC-generation oracle.

- A **MAC-generation** oracle takes as input a messages $M \in \mathcal{M}$. The oracle returns $T = F_K(M)$, the MAC of the queried string.
- A **MAC-verification** oracle takes as input a pair of strings $(M, T) \in \mathcal{M} \times \{0,1\}^\tau$. The oracle returns 1 if $\text{MAC}_K(M) = T$ and 0 otherwise.

See Figure 6.4. Note that, as the adversary asks its sequence queries, each query may depend on the results of prior ones. It is in this sense the adversary is *adaptive*.

The adversary's aim is to ask its MAC-verification oracle a query (M, T) that causes it to output 1 even though the query M was no previously made to the MAC-generation oracle. Such a pair (M, T) is called a *forgery*. The adversary wants to forge.

Equivalent to the above, one can say that the adversary's goals is to ask its MAC-verification oracle a query (M, T) that causes it to output 1 even though there was no previous query M to the MAC-generation oracle that produced a response of T. These notions equivalent, in our context of deterministic MACs, because, for any message M and key K, there's just one tag T that's going to cause the MAC-verification oracle to output 1.

We refer to the adversary's probability of finding a forgery as its *advantage*. We may write $\mathbf{Adv}^{\text{mac}}_F(\mathcal{A})$ for the probability that adversary $\mathcal{A}$ finds a forgery when it attacks the MAC F. The resources of importance are: the number q_{gen} of MAC-generation queries; the number q_{ver} of MAC-verification queries; and the adversary's running time t, usually assumed to include it's description size.

It should be acknowledged that the MAC definition given above is just one possible notion; there are notions that are stronger, weaker, or incomparable. Among these:

- One could focus on **key-recovery attacks** instead of forgery attacks. In some sense, key recovery "misses the point"—the purpose of MAC is not to make keys unrecoverable, but to make a message's origin determinable. The difference is seen mostly clearly in examples like the MAC $F_K(M) = 0^\tau$, an obviously insecure MAC whose key is never recoverable with probability exceeding $1/|\mathcal{K}|$. That said, *any* MAC can be forged on *any* message if the key can be recovered, so key-recovery attacks represent a particularly egregious way for a MAC to fail.
- One can restrict attention to **known-message** attacks instead of (adaptive) chosen-

message attacks. Here one requires that MAC-generation queries M_i are sampled uniformly from some specified distribution on strings. Standards like ISO 9797-1 attempt to distinguish, in discussing attacks, how many messages are "known" verses "chosen." The notion can be formalized, although rarely has it been. To allow both known and chosen messages, the model is like that in our definition of MAC security except that one would also support "sample" queries. Such a query returns a pair $(M, F_K(M))$ for a message M uniformly sampled from some fixed message space $\mathcal{X}$ associated to the attack.

- One could discard the MAC-verification oracle and make the adversary output a **single forgery attempt** (M, T). Equivalently, one can restrict the adversary to asking the MAC-verification oracle just $q_{\text{ver}} = 1$ query. This notion is actually the "original" one, going back to [17, 18] following [75]). See Bellare, Goldreich, and Mityagin for a discussion (reaching beyond deterministic MACs) of the relationship between this notion and our own [16].
- One can **deny credit for guessing** by numerically adjusting the advantage according to the tag length; one would set $\mathbf{Adv}^{\text{mac}}_F(A) = \Pr[\mathcal{A} \text{ forges}] - 2^{-\tau}$. Or one can go further and re-normalize advantage to the interval $[0, 1]$ by multiplying the entire right-hand side above $1/(1 - 2^{-\tau})$. We are not sure where such tag-length adjusted measures first appeared, but see [197] for profitable use of such adjusted accounting.

II.3. All PRFs are MACs; all <u>our</u> MACs, except GMAC, are PRFs. All secure and deterministic MACs considered in this report are not just good as MACs, but, also, good as *pseudorandom functions* (PRFs). Syntactically, a PRF looks just like a MAC: it is a map $F : \mathcal{K} \times \mathcal{M} \to \{0,1\}^\tau$. The game defining a PRF's security envisions placing the adversary into one of two environments. In the first, choose the key K at random from $\mathcal{K}$ and then give the adversary $\mathcal{A}$ oracle access to $F_K(\cdot)$. In the second, choose a uniformly random function ρ from $\mathcal{M}$ to $\{0,1\}^\tau$ and give the adversary oracle access to ρ. The adversary tries to predict in which environment it operates. One defines its *advantage* in doing this to be

$$\begin{aligned}\mathbf{Adv}^{\text{prf}}_F(\mathcal{A}) \;&=\; \Pr[K \xleftarrow{\$} \mathcal{K} :\; \mathcal{A}^{F_K(\cdot)} \Rightarrow 1] - \\ &\qquad \Pr[\rho \xleftarrow{\$} \text{Func}(\mathcal{M}, \{0,1\}^\tau) :\; \mathcal{A}^{\rho(\cdot)} \Rightarrow 1]\end{aligned}$$

the difference in the adversary's ability to output "1" in each of these two settings. Here $\text{Func}(X, Y)$ denotes the set of all functions from X to Y and endowed with the uniform distribution.

It is a simple and standard fact that if F is secure as a PRF then it is also secure a MAC. A quantitative formulation of this statement asserts that if there is an adversary $\mathcal{A}$ that breaks $F : \mathcal{K} \times \mathcal{M} \to \{0,1\}^\tau$ as a MAC then there is an adversary $\mathcal{B}$ that breaks F as a PRF, where $\mathbf{Adv}^{\text{prf}}_F(\mathcal{B}) \geq \mathbf{Adv}^{\text{mac}}_F(\mathcal{A}) - 2^{-\tau}$ and where adversary $\mathcal{B}$ runs in about the same time as adversary $\mathcal{A}$ and asks a single additional query. Note the additive $2^{-\tau}$ term. Even a "perfect" PRF will be forgeable with probability $2^{-\tau}$.

Because of the result above, when one is interested in analyzing a MAC one sometimes elects to analyze it as a PRF, instead. Effectively, one is showing more in this way—that the scheme is not *only* a good MAC, it is also a good PRF. It happens to be the case that all secure, deterministic MACs that have been analyzed in this report are secure as PRFs, and are analyzed in just this way.

II.4. Short tags. We mention one of the disadvantages of our notion for MAC-security: declaring the adversary successful as soon as it forges, the notion is simply not precise enough

to deal with the case of short tags (eg, $\tau = 8$), where one *expects* the adversary to occasionally forge, and may not be concerned about its doing so. As long as we are proving PRF-security, short outputs are definitionally treated just fine; if we switch to the MAC notion, however, the definition effectively fails to capture what one would want to say, that one can't forge more often than one "expects" to be able to forge.

II.5. FIL vs. VIL attacks. In describing our notions for MAC security we allowed the adversary to ask MAC-generation and MAC-verification queries on messages of any length. Similarly, for our PRF notion of security, the queries could have any length, as long as the messages were in the domain of the PRF. We can therefore term all of these adversaries as performing *variable input length* (VIL) attacks. One achieves VIL-security if one is immune to such attacks. We could, however, make a restriction and demand that all MAC-generation and MAC-verification queries use messages of the *same* length; or all queries to an adversary's PRF oracle have the same length. Such attacks would be called *fixed input length* (FIL) attacks. Achieving security with respect to this weaker class of attacks would be achieving FIL-security.

Why bother to think FIL-security when VIL-attacks are simple and natural, and defending against them is not hard? First, because there are simple MACs—the "raw" CBC-MAC being the outstanding example—that are secure in the FIL-sense but not in the VIL-sense. Second, it can be computationally cheaper to achieve FIL-security than VIL-security, and, sometimes, FIL-security is enough, because the receiver knows *a priori* how long valid messages must be. Finally, FIL PRFs can be used as a building block for making VIL PRFs, and this is often a reasonable strategy for their construction.

II.6. Ideal-cipher model for MACs. While we are primarily interested in the "standard" complexity-theoretic model for MACs, the fact is that many attacks implicitly assume the ideal-cipher model for stating their results. In addition, assertions like "there's an effective attack of 2^k time and $2^{n/2}$ queries"—or an assertion that there is no such attack—are best understood in the ideal-cipher model. As a consequence, we explain what the ideal-cipher model for blockcipher-based MACs entails, even though this model seems not, explicitly, to have ever been used.

The ideal-cipher model is often credited to Shannon [188], although this is a rather generous reading of his work. An explicit use of the model, in the setting of constructing a blockcipher rather than a MAC, is due to Even and Mansour [66], and then Kilian and Rogaway [110].

In our setting, the model would look like this. There is some construction $F\colon \{0,1\}^{k^*} \times \{0,1\}^* \to \{0,1\}^\tau$ for a MAC based on a blockcipher $E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$. We imagine instantiating the blockcipher with an ideal cipher—a uniformly chosen family of random permutations on n bits, each one named by a k-bit key K. A random key $K^* \in \{0,1\}^{k^*}$ is selected for the MAC, likely determining one or more keys for E. The adversary is given oracle access to $E(K,X)$, $E^{-1}(K,Y)$, and $F_{K^*}(M)$. The adversary's goal is, as usual, to forge. One looks at $\mathbf{Adv}^{\mathrm{mac}}_F(t, q_{\mathrm{gen}}, q_{\mathrm{ver}})$, the maximum probability of forgery over all adversaries that ask at most t queries to E or E^{-1} (total) and q_{gen} and q_{ver} queries to the MAC-generation and MAC-verification oracles for F.

The notion can be further refined, as in the complexity-theoretic setting, by distinguishing known-message queries and chosen-message queries. One can also switch the goal to key recovery, or to forging some randomly chosen message. In short, use of the model permits rigorous upper and lower bounds, including ones that speak about the impact of key length. Unfortunately, that has not been the tradition in this setting, and attempting to work out ideal-cipher-model results for different MACs is beyond the scope of this review.

The ideal-cipher model has been seen as moot for goals like message authentication since secure MACs are easily achieved in the standard model of computation. This thinking is wrong at two levels: ideal-cipher-model results would better reflect most of the described attacks, and ideal-cipher-model results would better capture key-length tradeoffs. We see the ideal-cipher model as a potentially useful adjunct to traditional provable-security analyses in this domain.

Chapter 7

CBC-MAC Algorithms 1–6

7.1. Summary. Document ISO/IEC 9797-1:1999 [91] (henceforth ISO 9797-1) is said to define some *six* different MAC algorithms, all CBC-MAC variants, referred to in the specification as MAC Algorithms 1–6. Each scheme takes as input a key and a string of essentially arbitrary length and produces a tag T. The standard should be praised for its carefully describing the desired security property [91, Section 3.2.6], which is existential unforgeability under and adaptive chosen-message attack, the security notion we defined in §II.2.

Many of the ISO 9797-1 schemes are of considerable historical importance. The standard follows but extends ANSI X9.9, ANSI X9.19, ISO 8731-1, and ISO/IEC 9797. Some algorithms in ISO 9797-1 enjoy widespread use, particularly in the banking sector.

In analyzing the ISO 9797-1 schemes the first issue to contend with is that the number of modes defined in the spec is not actually six, but considerably more. This is because the modes are parameterized not only by the underlying blockcipher $E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ and tag length τ, where $1 \le \tau \le n$, but on one or two further parameters: the *padding method* and, in some cases, implicitly, the *key-separation method*, too.

In Figure 7.1 we summarize the status of the various ISO 9797-1 MACs. Much of this chapter revolves around explaining the contents of this table. A check mark (✓) in column "Pf" of this table means that we regard the mode as having provable-security results that substantially establishes the scheme's goals; the meaning of "provably secure" varies here according to what *are* the scheme's goals. The key goals are summarized in column "Goals."

Considering the long history of the MACs in ISO 9797-1, our overall assessment is not very positive. To begin with, there are simply too many schemes. Several of them have been shown not to achieve their cryptographic objectives. Other schemes have no provable-security results in support of their design aims, or else have provable-security results that only partially establish what is wanted. There are no proofs in evidence of constructions for enhanced key-length, while the first provable-security result for beyond-birthday-bound security has only just arrived. Some of the schemes target an overly weak cryptographic goal—VIL security—an invitation to mechanism misuse. Beyond all this, the motivation underlying the design and selection of the schemes is somewhat out-of-date: while it would be an overstatement to assert that 64-bit blockciphers are dead, the prevalence of 128-bit designs, particularly AES, and diminishing support for 64-bit blockciphers, particularly the no-longer-FIPS-approved DES, combine to make enhanced-key-length or beyond-birthday-bound designs of marginal practical utility. Finally, the ISO 9797-1 schemes were simply not designed with a provable-security viewpoint in mind, leading not only to some bad schemes and to problems in how key-separation is envisioned to work. All of these issues conspire to create a standard that comes across rather

Alg	Pad	Sep	Klen	Mlen	#Calls	Goals	Proof	Attack	Ref
1	1	—	k	$[0..\infty)$	$\lceil \mu/n \rceil$	B F	✓	—	[18]
1	2	—	k	$[0..\infty)$	$\lceil \mu'/n \rceil$	B V	—	✓	folklore
1	3	—	k	$[0..2^n-1]$	$\lceil \mu/n \rceil + 1$	B V	✓	—	[167]
2	1	opt	k, $2k$	$[0..\infty)$	$\lceil \mu/n \rceil + 1$	B F	✓	—	[167]
2	2	opt	k, $2k$	$[0..\infty)$	$\lceil \mu'/n \rceil + 1$	B V	✓	—	[167]
2	3	opt	k, $2k$	$[0..2^n-1]$	$\lceil \mu/n \rceil + 2$	B V	✓	—	[167]
3	1	—	$2k$	$[0..\infty)$	$\lceil \mu/n \rceil + 2$	B F K	✓	✓	[18]
3	2	—	$2k$	$[0..\infty)$	$\lceil \mu'/n \rceil + 2$	B V K	✓	✓	[39]
3	3	—	$2k$	$[0..2^n-1]$	$\lceil \mu/n \rceil + 3$	B V K	✓	✓	[167]
4	1	✓	$2k$	$[n+1..\infty]$	$\lceil \mu/n \rceil + 2$	B F K	✓	✓	[55]
4	2	✓	$2k$	$[n..\infty]$	$\lceil \mu'/n \rceil + 2$	B V K	✓	✓	[55]
4	3	✓	$2k$	$[0..2^n-1]$	$\lceil \mu/n \rceil + 3$	B V K	✓	✓	[54]
5	1	✓	k	$[0..\infty)$	$2\lceil \mu/n \rceil$	C F	—	✓	[105]
5	2	✓	k	$[0..\infty)$	$2\lceil \mu'/n \rceil$	C V	—	✓	[105]
5	3	✓	k	$[0..2^n-1]$	$2\lceil \mu/n \rceil + 2$	C V	—	—	—
6	1	✓	$2k$	$[n+1..\infty]$	$2\lceil \mu/n \rceil + 4$	C F K	✓	—	[206]
6	2	✓	$2k$	$[n..\infty]$	$2\lceil \mu'/n \rceil + 4$	C V K	✓	—	[206]
6	3	✓	$2k$	$[0..2^n-1]$	$2\lceil \mu/n \rceil + 6$	C V K	✓	—	[206]

Figure 7.1: **MAC Algorithms of ISO 9797-1**. The six algorithms (**Alg**) depend on blockcipher $E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$, tag length τ, and padding method $i \in \{1,2,3\}$ (**Pad**). Column **Sep** indicates if the mode depends on a key-separation algorithm (opt for optional). Columns **Klen** and **Mlen** give the key length and permitted message lengths. Column **#Calls** gives the number of blockcipher calls to process a μ-bit string. Here $\mu' = \mu+1$. Column **Goals** defines the "intended" security goals: B for provable security up to the birthday bound; C for provable security *beyond* the birthday bound; F for proofs in the FIL (fixed input length) setting; V for proofs in the VIL (variable input length) setting; K for enhanced key-length security. Column **Pf** indicates if there are provable-security results to evidence the specified goals—excluding goal K, for which no proofs are known. In the presence of key-separation, some of the provable-security results would be in the ideal-cipher model, or would assume an all-distinct-keys variant. Column **Atk** indicates if there are "damaging" attacks. This is inherently subjective, and not truly binary; for example, I indicate that there are attacks on all of Algorithms 3 and 4 because one can recover the key with time complexity substantially less than one would expect—not because the attacks are necessarily practical. The attacks vary in effectiveness. Column **Ref** gives the most important reference.

as a relic from another age, even though it's not so very old.

For the reasons just described, I recommend against blanket adoption of the ISO 9797-1 schemes. Considerably fewer options should be supported for MAC Algorithms 1–4, if any of these modes are to be supported at all, while MAC Algorithms 5–6 should be avoided as specified. The "best" algorithms in the too-big basket of ISO 9797-1 MACs are: MAC Algorithm 1 with padding schemes 1 or 3; the two-key version of MAC Algorithms 2 with padding scheme 2 and two independent keys; MAC Algorithm 3 with padding scheme 2; and a 6-key version (which is not actually permitted in the spec) of MAC Algorithm 6 with padding scheme 2. For each of these algorithms, more than with the rest, one could argue for inclusion based on simplicity, widespread usage, historical importance, cryptanalytic history, or provable-security results.

Alternatively, one doesn't have to go the route of picking and choosing among the many ISO 9797-1 CBC-MAC variants; one could reject the batch in toto, in spite of their historical importance. The *raison d'être* for all this complexity is, fundamentally, to overcome the small blocksize and short key length of DES. The extent of support for ISO 9797-1 schemes would probably depend on how committed on is to legacy 64-bit blockciphers and modes.

Overall, the CMAC algorithm represents a cleaner choice as a CBC-MAC variant, and it does all that a CBCMAC-variant should *if* one is assuming a modern 128-bit blockciphers underneath. What that algorithm does not do is to provide good protection when the key length or block length are short.

7.2. The ISO/IEC 9797-1:2010 final draft. A few days before this report was due I learned from Bart Preneel that an effort has been underway, since 2007, to revise ISO/IEC 9797-1:1999. The final draft of the revision, ISO/IEC FDIS 9797-1:2010 [93], has just been approved, so it is likely that the revised standard will be published this year. The main changes in 9797-1 are as follows:

1. MAC Algorithms 5 and 6 have been eliminated (beyond-birthday-bound security is no longer a supported goal for any of the 9797-1 schemes);

2. A new MAC Algorithms 5 has been added. It is the CMAC algorithm (see Chapter 8).

3. A new MAC Algorithm 6 has been added. It is a scheme similar to but more efficient than MAC Algorithms 2 and 3 (one just switches keys for the final blockcipher call).

4. Key separation is handed better than before. It is much more explicit than before, and a conventional and provable-security-friendly method (an approach using a master key, CTR mode, and a blockcipher) has now been pinned down.

It is beyond the scope of this review to provide a proper evaluation of the updated ISO standard. Revision ISO/IEC FDIS 9797-1:2010, which has been available to ISO members since at least 2010-12-01, might well have been a better target for my evaluation than the 1999 edition of the standard. But I learned of the coming edition of the standard too late to suggest substituting it for my evaluative target.

7.3. Some surveys. Quite a bit has been written on attacks of the various MAC Algorithms of ISO 9797-1 (rather less has been written on relevant provable-security results). We point the reader to Brincat and Mitchell [47] and Joux, Poupard, and Stern [105] for nice summaries of existing attacks (as well as those authors' new results). See too the lecture slides from a talk by Bart Preneel [169].

7.4. New notation. We now move on to describing the schemes of the ISO standard. While the spec is not long—a total of 16 pages, with all the algorithms defined in just the first six—it is not described along conceptual lines or notation that we like. (For example, the key-derivation maps that are crucial to some of the techniques are never explicitly named; the initial transformation would be better conceptualized *before* the first block is processed; and we don't like notational choices that include calling the underlying blockcipher e (and sometimes "e"), using H_q and G for strings, and so on. What we describe in this chapter is fully equivalent to what is in the specification document, but we have tried to clean it up, produce nice diagrams, and so on.

7.5. Padding methods. Each of the ISO 9797-1 algorithms depends on a *padding method*. One can consider the padding method as a parameter of the schemes, although changing this parameter so changes the nature of the scheme—including its domain—that one may prefer to think of each padding method as determining a fundamentally *different* scheme.

Fix the blocklength n of the underlying blockcipher. We will refer to the three padding methods as Pad_1, Pad_2, and Pad_3. Each is a map from strings to strings. Each of these maps may be used with each of the six specified MAC algorithms, creating, it would seem, some 18 different MAC schemes. (We will see soon why this remains an under-accounting.) The three padding methods are defined as follows:

7.5.1 Pad_1 is the 0^*-appending padding technique. The message M is mapped to $M\,0^i$ where i is the least nonnegative number such that $|M| + i$ is a positive multiple of n.

7.5.2 Pad_2 is the 10^*-appending padding technique. The message M is mapped to $M\,10^i$ where i is the least nonnegative number such that $|M| + i + 1$ is a positive multiple of n.

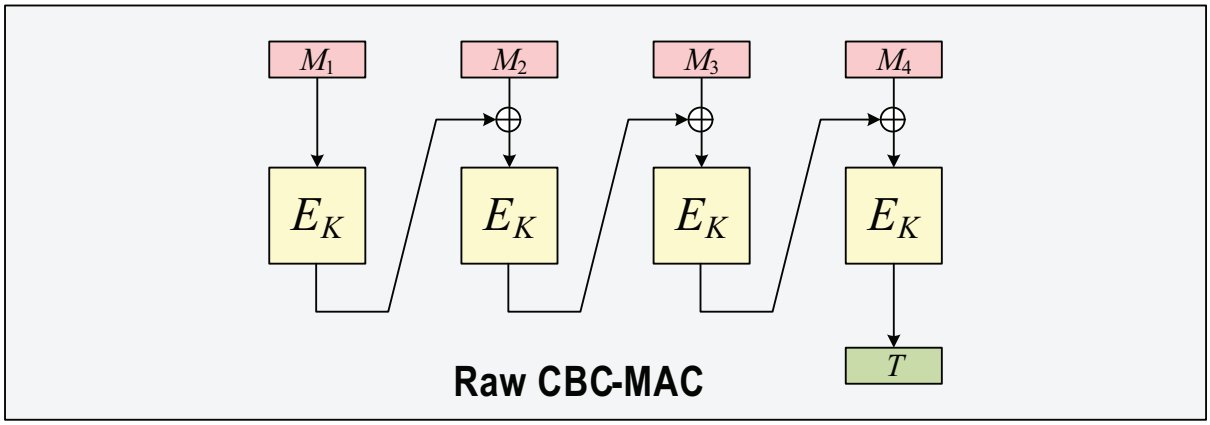
7.5.3 Pad_3 is the length-prepending padding technique. The message M is mapped to $L\,M\,0^i$ where i is the least nonnegative number such that $|M| + i$ is a positive multiple of n and where L is the binary encoding of $|M|$ into a field of n bits. To make the encoding of $|M|$ fit into L one insists for Pad_3 that the message M satisfy $0 \le |M| \le 2^n - 1$.

The idea is that one *first* applies the padding method and *then* applies the MAC Algorithm. The composite mechanism is the actual MAC, the object interest.

7.6. Truncation. Each of the ISO 9797-1 MAC algorithms depends on a blockcipher $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ and a padding method Pad from among those in $\{\mathrm{Pad}_1, \mathrm{Pad}_2, \mathrm{Pad}_3\}$. Beyond this, each mode is parameterized by a *tag length*, an integer $\tau \in [1\,..\,n]$. The output of all schemes is truncated to this number of bits (take the *first*, or most significant, τ bits). In short, the technique is to: (i) pad, (ii) apply the "core" algorithm (that is, Algorithm 1–6), and then (iii) truncate the output to τ bits.

The purpose of truncation is normally imagined to be the reduction of communications complexity; one sends 64-bits instead of 128 bits, say, because it saves eight bytes and a forging probability of 2^{-64} is, usually, quite adequate. All ISO 9797-1 MACs are, presumably, PRFs, and truncation of a PRF gives a PRF that is, with respect to the $\mathbf{Adv}^{\mathrm{prf}}$-notion, just as good. In other words, truncation of a MAC to τ bits never harms the PRF-security *already* enjoyed by a construction beyond the $2^{-\tau}$ risk of forgery that one inherently incurs by the adversary simply guessing a τ-bit tag.

7.7. Key-separation. As a final parameter, some of the ISO 9797-1 MAC Algorithms employ a *key-separation algorithm*, which we denote by Sep. We will describe later what are the inputs and




```
00  algorithm CBCMAC_K(M)                                   Raw CBC-MAC
01  if |M| is not a positive multiple of n then return Invalid
02  M_1 ··· M_m ← M where |M_i| = n
03  C_0 ← 0^n
04  for i ← 1 to m do C_i ← E_K(M_i ⊕ C_{i−1})
05  return C_m
```

Figure 7.2: **The raw CBC-MAC.** The algorithm is parameterized by an n-bit blockcipher E. The string being processed must have mn bits for some $m \geq 1$.

outputs to these key-separation maps. Key separation is an area where the standard is especially bad: the key-separation maps are never even named, nor are their significant cryptographic properties specified. The omission means that the standard's MAC Algorithms 4, 5 and 6, as well as the key-length-k version of MAC Algorithm 2, are all underspecified; one can't, for example, publish test vectors for any of these modes, since no definitive method is indicated to produce the internal keys. Beyond this, the example key-separation methods described in the standard would destroy any blockcipher-based provable-security claims, as they depend on *ad hoc* ways to derive keys, ways that interact badly with the PRF and PRP notions of security, which do not guarantee unrelated maps E_K and $E_{K'}$ for related but distinct keys K and K'.

7.8. Raw CBC-MAC. We have not yet defined any of the ISO 9797-1 MACs. Before we do so, it is useful to define and review the MAC from which they spring—what we call the "raw" CBC-MAC.

The raw CBC-MAC, which we also denote as CBCMAC, is specified in Figure 7.2. The algorithm is classical, underlying the techniques described not only in ISO 9797-1 but also ANSI X9.9, ANSI X9.19, FIPS 81, FIPS 113, ISO 8731-1, ISO 9807, and ISO 9797. The raw CBC-MAC can be considered as a derivative of the CBC encryption scheme (Chapter 4): the message M that we wish to MAC is CBC-encrypted, but only the *final* block of ciphertext is returned as the MAC.

While useful and classical, the raw CBC-MAC has some major restrictions, some of which underlie the many choices offered in ISO 9797-1. We single out five different issues:

7.8.1 **Restricted domain.** The input to the raw CBC-MAC must be a positive multiple of n bits, where n is the blocklength of the underlying blockcipher $E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$; the algorithm is *not defined* for other input lengths. Since real-world messages may have other lengths, for a generally-useful standard the raw CBC-MAC is combined with additional technique—for example, padding techniques like those defined before—

to make it fully usable.

7.8.2 **Cut-and-paste attacks.** When the adversary may ask messages of varying lengths, the raw CBC-MAC is simply *wrong*: it is easy for the adversary to forge. As a simple example, suppose the adversary obtains $T = \mathrm{CBCMAC}_K(M)$ for some arbitrary single-block M. Then the adversary can forge the message $M \,\|\, (M \oplus T)$ with a tag of T. Recalling the discussion from §II.5 we say that the raw CBC-MAC is not *VIL* (variable input length) secure; it is only *FIL* (fixed input length) secure. What is fixed or variable is the length mn of queries the adversary may ask—including the length mn of its forgery attempt.

7.8.3 **Birthday attacks.** Even if we fix all strings to have the *same* message length, there will be birthday attacks on the CBC-MAC, meaning attacks that need about $\sqrt{N}$ queries, where $N = 2^n$, to forge a string. This is not much of a problem when the underlying blockcipher is AES, with its $n = 128$ bit blocks, but it is a potential problem when the blockcipher is DES, or some other $n = 64$ bit blockcipher. Birthday attacks are not limited to the raw CBC-MAC; they arise in any CBCMAC-like mode where one chains ones way through a sequence of blocks using an n-bit blockcipher; Preneel and van Oorschot [172] and Preneel [171]. Recent birthday-bound attacks on CBCMAC-like constructions are more damaging still [102], since they allow selected forgeries.

7.8.4 **Key-guessing attacks.** When k is the length of the underlying blockcipher, the underlying key for the raw CBC-MAC can be recovered—and arbitrary messages then forged—if the adversary spends 2^k time and has a handful of plaintext/ciphertext pairs (at least n/k, say). This is not much of a problem when the underlying blockcipher is AES, with its keys of $k \geq 128$ bits, but it is a problem when the blockcipher is DES, or some other blockcipher with an overly short key.

7.8.5 **Efficiency issues.** First, the number of blockcipher calls needed to MAC a message M is proportional to the message length—$|M|/n$ calls are used. Second, these calls must be computed serially, one after another. The latter characteristic can slow down modern processors, not just dedicated crypto-hardware. Neither restriction is essential to a blockcipher-based MAC. In some applications, either of these efficiency restrictions can be problematic.

In response to these concerns,

7.8.1* All algorithms in ISO 9797-1 aim to overcome the limited-domain restriction. They do so by beginning the message authentication process with a mandatory padding step, using one of the three mechanisms Pad_1, Pad_2, or Pad_3.

7.8.2* Many of the ISO 9797-1 schemes aim to overcome cut-and-paste attacks. We will assert that the "intent" of using Pad_2 or Pad_3 instead of Pad_1 includes overcoming this attack and achieving VIL security. Use of type-1 padding leads to, at best, FIL security over $\{0,1\}^*$ since strings M and $M\,0^i$ will have identical MACs as long they have the same number of blocks, meaning $\lceil |M|/n \rceil = \lceil (|M| + i)/n \rceil$. This was understood to the ISO 9797-1 designers, as made clear in [91, Annex B].

7.8.3* The designers of ISO 9797-1 thought that their MAC Algorithms 5 and 6 would overcome birthday attacks [91, Annex B]. We will see shortly that this belief was false for MAC Algorithm 5 (there is an attack by Coppersmith, Knudsen, and Mitchell [54, 55]) and true for MAC Algorithm 6 (there is a recent proof by Yasuda [206]).

7.8.4* The designers of ISO 9797-1 thought that their MAC Algorithms 4 and 6 would over-

come key-guessing attacks [91, Annex B]. For MAC Algorithm 4, we will see, this has been shown to be false [54, 55]. For MAC Algorithm 6 the question remains open; no attacks or provable-security results exist to support or rule-out the conjecture. A random-oracle-model result ought to be possible in this case.

7.8.5* The designers of ISO 9797-1 did *not* try to overcome the CBCMAC efficiency issues (lack of parallelizability and proportional-to-message-length blockcipher use). This makes sense; overcoming these limitations would take one outside the realm of any CBCMAC-like scheme.

7.9. Key provable-security results. The following results underlie some of the comments we make on the ISO 9797-1 schemes. While we will repeat some of this material when the relevant schemes are defined, it seems desirable to bring together in one place the most significant provable-security results on the CBCMAC and CBCMAC-like schemes that arise in ISO 9797-1.

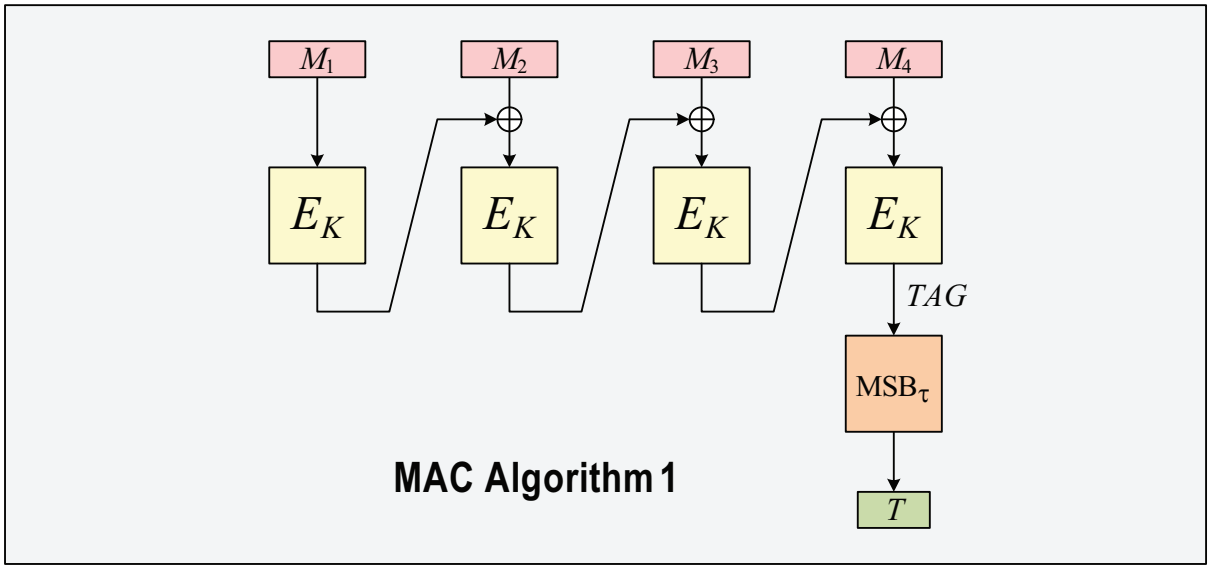
7.9.1 Bellare, Kilian, and Rogaway [18] have proven the FIL-security of the raw CBC-MAC The quantitative theorem statement shows that an adversary that asks q queries, each having a length of m blocks, can forge with probability that is bounded above by $2^{-\tau} + 2q^2m^2/2^n$ plus the insecurity of the underlying blockcipher, $\mathbf{Adv}^{\mathrm{prp}}_E$, as a random permutation. (Here and elsewhere, we sometimes assume a single forgery attempt.) What is actually demonstrated of the raw CBC-MAC is security as a (fixed-input-length) PRF, not just as a MAC.

7.9.2 Petrank and Rackoff [167] extended the results above to show that the raw CBC-MAC retains its provable-security in the VIL setting if the message space is *prefix free*. Saying that the message space is prefix-free means is that the adversary may not ask messages M and M' where one is a prefix of the other. One way to obtain a prefix-free encoding is the length-prepending method Pad_3. There are other prefix-free encoding-schemes too. One problem with the length-prepending approach is that it is not *on-line*: one needs to know the length of M before one can begin the process of MACing it. Petrank and Rackoff also prove security for the *encrypted* CBCMAC, "EMAC," where the CBCMAC result in enciphered using the same blockcipher with a separate and independent key. As we will see, this is essentially MAC Algorithm 2 of ISO 9797-1. The encrypted CBCMAC *is* online. For both Petrank-Rackoff results, security is in the VIL PRF-sense, and it falls off in $\sigma^2/2^n$ plus the PRF-insecurity of the underlying blockcipher. Here σ is the total number of n-bit blocks asked by the adversary.

7.9.3 Black and Rogaway [39] proved VIL security for CBCMAC variants that include MAC Algorithms 2 and 3 with padding method 2. Their work also simplifies the proof of Petrank and Rackoff [167], connecting the method used in MAC Algorithms 2 and 3 to Carter-Wegman universal hashing [50, 201].

7.9.4 Nandi [149], following earlier work by Bellare, Pietrzak, and Rogaway [24], Bernstein [29], and Nandi [148], has provided the best bounds to date for the CBCMAC: on a prefix-free message space, an adversary that asks q queries that total σ blocks will achieve PRF-advantage of at most $11\sigma q/2^n$ provided each query is of $2^{n/3-1}$ or fewer blocks.

7.9.5 Pietrzak [168], following Bellare, Pietrzak, and Rogaway [24], provides the best bound on MAC Algorithm 2 (EMAC), bounding an adversary's PRF-advantage by $q^2/2^n$, when messages are of $m \leq 2^{n/8}$ blocks. The best prior bound had a low-order dependency on m, taking the form $q^2m^{o(1)}/2^n$.



MAC Algorithm 1

10 **algorithm** $\text{ALG1}_K(M)$
11 $M \leftarrow \text{Pad}(M)$
12 $Tag \leftarrow \text{CBCMAC}_K(M)$
13 $T \leftarrow \text{MSB}_\tau(Tag)$
14 **return** T

Figure 7.3: **MAC Algorithm 1.** The scheme ALG1 depends on parameters Pad, E, and τ. Our definition employs the raw CBC-MAC, as defined in Figure 7.2.

7.9.6 Yasuda [206] has recently provided the first beyond-the-birthday bound provable-security results for a CBCMAC-like scheme. He focuses his attention on a MAC that consists of the xor of two independent copies of EMAC. But, as Yasuda himself points out, the results also apply to MAC Algorithm 6 (with any form of padding).

Conspicuously missing from the provable-security results on the CBC-MAC is anything demonstrating that there is cryptographic value in truncation. Also absent is any result on the effective key-length of CBCMAC-like schemes—results that, presumably, would be in the ideal-cipher mode.

7.10. Bracket-notation for attacks. Appendix B of ISO 9797-1 employs a compact notation to summarize the efficiency of an attack on a MAC: one writes a bracketed four-tuple $[t, a, b, c]$ where t represents the running time, a represents the number of known message/MAC pairs, b represents the number of chosen message/MAC pairs, and c represents the number of on-line verification queries. A good deal of follow-on work has adopted this same notation, or simplifications of it. The notation is never formally defined, although it certainly could be if one assumes an ideal-cipher model and defines separate known, chosen, and verification queries for the adversary. We will occasionally use a simplified version of the notation, retaining just $[t, a + b + c]$: time and number of queries. To make this rigorous, one would assume the ideal-cipher model and say that a $[t, q]$ attack asks at most t blockcipher queries, q MAC-generation queries, and achieves forging advantage of at least, say, $1/4$.

7.11. MAC Algorithm 1. We are now ready discuss each of the ISO 9797-1 schemes in turn. This first algorithm, the "basic" CBC-MAC, is the raw CBC-MAC except for the inclusion of padding at the beginning and truncation at the end. See Figure 7.3. The method seems to begin

with ANSI X9.9 (1982 for the original version) [4], where $E = \text{DES}$, $\tau = 32$, and $\text{Pad} = \text{Pad}_1$.

Assuming that E is a PRP, the VIL security of ALG1 depends on the choice of Pad and τ. We recall first that $\text{Pad} = \text{Pad}_1$ will never give rise to a VIL-secure scheme, as already described; messages 0 and 00, for example, will have identical MACs. The value of τ is not relevant.

Let's assume $\tau = n$, the blocksize of E. One might be tempted to think that Pad_2 may fare better than Pad_1, as no two messages pad to the same string. This would be wrong. As an example, let the adversary query $(n-1)$-bit strings until it finds one, M, where the tag $T = E_K(M \,\|\, 1)$ ends in a 0-bit. The adversary will need just two expected trials. Then the adversary can forge a tag for $M \,\|\, 1 \,\|\, T[1\,..\,n-1]$, as the MAC of this string will once again be T.

If τ is sufficiently small, like $\tau = n/2$, then the simple attacks we have just described on ALG1/Pad_2 do not work. But Knudsen has shown [113] that truncation doesn't help nearly as much as one would hope; there remains an attack of complexity $2^{(n-\tau)/2}$ queries. Concretely, using MAC Algorithm 1 with a 64-bit blockcipher (say DES or triple-DES) and truncating to 32-bits will admit forgeries with just 2^{17} queries. Using the (streamlined) bracket notation, one would assert that there's a $[0, 2^{17}]$ attack. In short, while truncation doesn't "damage" a PRF, it does not, as one might hope, transform this particular FIL-secure PRF into a just-as-sound VIL-secure one.

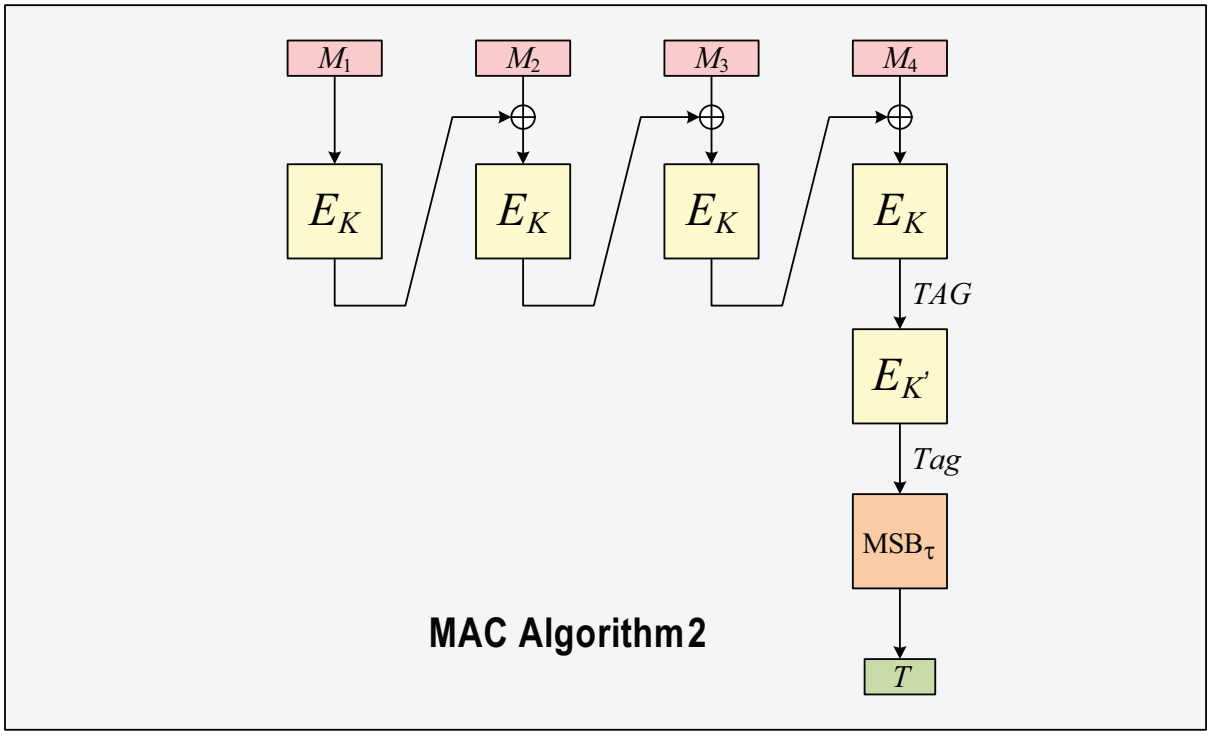
Use of ALG1 with padding Pad_3 does give rise to a VIL-secure MAC. This is an instance of the Petrank and Rackoff result [167]: length-prepending is a prefix-free encoding.

The provable security results for MAC Algorithm 1 (FIL-security with padding method-1, or VIL-security with padding-method 3) go only as far as the birthday bound: the PRP-advantage falls off in $\sigma q/2^n$ if the adversary asks σ blocks worth of queries, meaning that there is *no provable-security assurance* at around $2^{n/2}$ queries (or 2^k time) [18, 149]. These bounds are tight: there are simple "birthday attacks" on MAC Algorithm 1 (padding methods 1 or 3) that forge with about $2^{n/2}$ blocks. Indeed this is true for any iterated MAC with an n-bit pipe. Such attacks are described, for example, by Preneel and van Oorschot [171]. The idea is easy. Assuming still that $\tau = n$, let the adversary ask three-block messages, for example, with a fixed third block and random blocks 1 and 2. When a collision is found, $\text{MAC}_K(ABC) = \text{MAC}_K(A'B'C)$, we will also have that $\text{MAC}_K(ABD) = \text{MAC}_K(A'B'D)$ for any D. So asking for the MAC of ABD lets one forge the MAC for $A'B'D$. More sophisticated and productive birthday attacks also exist. See [102], for example, for recent work in this direction.

It is unfortunate that ALG1's basic security properties depend in a complicated way on the values of Pad_i and τ, even assuming that the blockcipher E is a good PRP. One cannot make simple statements, like "ALG1 is a good PRF assuming the underlying blockcipher is a good PRP" since the statement is not always true.

7.12. MAC Algorithm 2. The schemes—we regard there as being *two*—are defined in Figure 7.4. In the more "basic" variant, ALG2A, the MAC key has $2k$ bits; keys are of the form $K \,\|\, K'$ where $K, K' \in \{0,1\}^k$ are keys for the underlying blockcipher. The key K' is used to encrypt the raw CBC-MAC prior to truncation. Mode ALG2B is identical except that some key derivation function, which we call Sep, defines K' from K.

MAC Algorithm 2 with type-1 padding emerged during the EU's RACE project, where, when used with DES, it is named the RIPE-MAC1 [45]. Petrank and Rackoff [167] studied the variant of ALG2A with no padding and restricted to strings that are a multiple of n bits. They called the method EMAC, for the Encrypted (CBC-)MAC. When Black and Rogaway looked at the scheme, they renamed it ECBC [38], as three names weren't yet enough.



20 **algorithm** $\text{ALG2A}_{K\,K'}(M)$ *MAC Algorithm 2A*
21 $M \leftarrow \text{Pad}(M)$
22 $TAG \leftarrow \text{CBCMAC}_K(M)$
22 $Tag \leftarrow E_{K'}(TAG)$
24 $T \leftarrow \text{MSB}_\tau(Tag)$
25 **return** T

26 **algorithm** $\text{ALG2B}_J(M)$ *MAC Algorithm 2B*
27 $K \parallel K' \leftarrow \text{Sep}(J)$
28 **return** $\text{ALG2.2}_{K\,K'}(M)$.

Figure 7.4: **MAC Algorithm 2.** Parameters are Pad, E, and τ. In variant-A (my terminology) the underlying key is a pair of k-bit strings K, K' where the blockcipher is $E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$. In variant-B there is an additional parameter, Sep, and $K\ K'$ is determined by applying this to the k-bit key J.

Let us begin with ALG2A. As always, VIL security is not achieved with Pad_1 padding. But it *is* achieved (up to the birthday bound) with Pad_2 padding. The result follows from the Petrank and Rackoff result [167], which was quantitatively improved by Bellare, Pietrzak, and Rogaway [24] and Pietrzak [168]. Even though those authors assumed all messages were a multiple of n bits and no padding is used, it is easy to establish that that any PRF that is VIL-secure on $(\{0,1\}^n)^+$ will become a VIL-secure PRF on all of $\{0,1\}^*$ when composed first with Pad_2. The reduction is tight (meaning there is no significant loss in quantitative provable security).

Simplifying the Petrank-Rackoff proof, it is easy to see *why* ALG2A is VIL-secure to the birthday bound. The idea is contained in Black and Rogaway [38]. In the proof one is working in the information-theoretic setting, passing to the complexity-theoretic setting only at the end. So mentally replace each E_K in Figure 7.4 by a random permutation π, and mentally replace $E_{K'}$ by an independent random permutation π'. Conceptually group together all that

goes on in the CBCMAC portion of the scheme. One establishes that it, the raw CBC-MAC over a random permutation, is an *almost-universal hash-function*. Here this means that, for $M \neq M'$, $\Pr_\pi[\mathrm{CBCMAC}_\pi(M) = \mathrm{CBCMAC}_\pi(M')]$ is small—about $m^2/2^n$. Here M and M' have m or fewer blocks and the probability is taken over a random n-bit to n-bit permutation π. One then observes that enciphering the output of *any* almost-universal hash-function with an independently chosen random permutation makes a good PRF—and hence a good MAC. This is the basic idea of the Carter-Wegman approach to making a MAC [50, 201] (although those authors did not imagine using a blockcipher; they thought to use a one-time pad).

As for the use of Pad_3 padding in MAC Algorithm 2, this would seem to be a poor choice, insofar as the extra blockcipher call at the end is already designed to provide VIL security. Still, since one obtains VIL PRF-security without the extra blockcipher call using type-3 padding (see the discussion of MAC Algorithm 1), its inclusion does nothing to compromise the VIL PRF property; with type-3 padding, MAC Algorithm 2 would be secure (to the birthday bound) even were K' made public.

As for ALG2B, a fundamental question to ask is what property the key-separation algorithm Sep needs to have. The ISO standard says nothing in this direction—and what it does say goes against achieving provable-security guarantees, and is even self-contradictory. The spec begins by saying [91, p. 5, column 1] that the value of K' (inexplicably named K''' in the original spec) may be derived from K in such a way that the two are different. This would suggest that line 27 of Figure 7.4 misrepresents the mode's key separation; it should say $K' \leftarrow \mathrm{Sep}(K)$, with K, not some (invented) J, as the key at line 26. But [91, Section 7.2, note 1] goes on to say that another way to do the key-separation is to derive both K and K' from a common master key. As this way is more general—and more closely hits the mark on what one *should* do—we have generously elevated this remark into a description of how ALG2B works.

Also in [91, Section 7.2, note 1] there is a concrete suggestion illustrating the sort of key separation the authors have in mind: they say to let $K = L$ and let K' be L after complementing every other run of four bits, beginning with the first. If key separation like this is used, there will be no chance for a provable-security guarantee regardless of the padding method used—not under the PRP assumption on E. This is because we will produce related keys and the PRP-assumption says nothing about how the blockcipher E will behave on related keys—one may well have related behaviors. To be concrete, assume we have a PRP-secure blockcipher $E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$. From it one can define the blockcipher $E'\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ by asserting that $E'_K(X) = E_K(X)$ if the first bit of K is 0, and $E'_K(X) = D_{K'}(X)$ if the first bit of K is 1, where K' is K with every other run of four bits complemented, beginning with the first. It is easy to see that if E is a secure PRP then so is E'. But using E' in the suggested construction is trivial to break, since the final pair of encryptions, $E'_K \circ E'_{K'}$, is just the identity map.

A better approach for key separation would have been to use a more standard and provable-security-friendly key-separation technique, like $K = E_J(C_1)$ and $K' = E_J(C_2)$ for distinct n-bit constants C_1 and C_2. Use of such a technique would allow one to claim provable security for ALG2B under the corresponding conditions on ALG2A.

One can also obtain provable-security results even for *ad hoc* techniques like the spec's bit-flipping method if one switches to regarding the blockcipher as an *ideal* cipher—in other words, if one switches to the ideal-cipher model. But one should not need to be making such a strong assumption (or model) in the setting of providing a secure MAC. We believe that the designers of the ISO 9797-1 spec often *did* regard the underlying blockcipher as ideal. One sees this, for example, in statements like "It is assumed that the block cipher has no weaknesses" [91, p. 13, column 1], a sentence that is hard to make sense of except to say that one is thinking in terms

of the ideal-cipher model.

Note that, instead of moving to the ideal-cipher model in the presence of key-separation, one could remain in the blockcipher-as-a-PRP world but model the key separation *itself* as a random oracle. The problem with this viewpoint is that the suggested key-separation schemes are clearly *not* RO-like: one flips some bits, say, rather than applying a hash function.

The under-specified and *ad hoc* key-separation mechanisms of ISO 9797-1 are a place where once sees quite clearly the absence of a provable-security informed design.

The benefit MAC Algorithm 2 compared to MAC Algorithm 1 is to achieve VIL-security when combined with type-2 padding. Type-2 padding is preferable to type-3 padding not only because it saves a blockcipher call but, also, because it is on-line: one does not need to know the length of the message M in advance before beginning to compute its MAC. Where the spec goes wrong is in allowing too many additional options—type-1 padding, type-3 padding, and key separation.

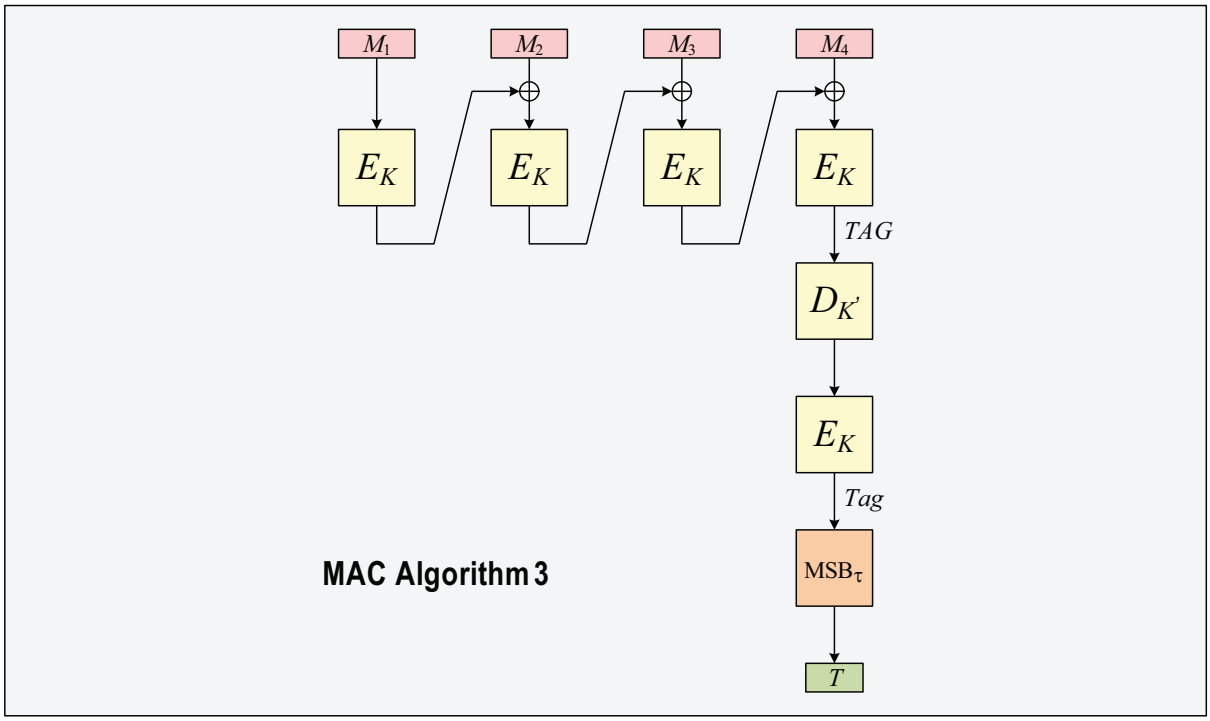
Provable-security to the birthday bound [24] is once again tight; regardless of the padding method used; there are known attacks that employ q queries to get forging advantage of about $q^2/2^n$. Such attacks are described, for example, by Preneel and van Oorschot [171].

If one is using DES as the underlying blockcipher, or any other blockcipher with a too-short key, then key-guessing attacks will continue to be an issue with MAC Algorithm 2, despite its option for a length-$2k$ key. With the mode, using a standard meet-in-the-middle attack and assuming $\tau = n$, one can recover the length-$2k$ key in 2^k time, using just one message/tag pair. The attack is classical (it's essentially the meet-in-the middle attack of Merkle and Hellman [140]); see Mitchell [145] for some discussion. Nor does truncation help; for example, there remains a $[0, 2^{34}]$ attack when $n = 64$ and $\tau = 32$ [169]. The existence of a meet-in-the-middle attack on MAC Algorithm 2 helps explain the existence of MAC Algorithm 3.

7.13. MAC Algorithm 3. This method dates to ANSI X9.19 [3] (1986), a standard for retail banking; consequently, the blockcipher mode has been called the "retail MAC." It is like MAC Algorithm 2 but, rather than "seal" the last block by enciphering the raw CBC-MAC, the designers instead switch to triple encryption for sealing it. Key-separation is no longer a permitted option; the underlying key *must* be $2k$ bits. See Figure 7.5.

Assuming the underlying blockcipher is a secure PRP, MAC Algorithm 3 is VIL-secure, again to the birthday bound, assuming type-2 padding. To explain why this is so we again consider the variant of the algorithm that omits padding and correspondingly restricts the domain to $(\{0,1\}^n)^+$. As commented on earlier, VIL-security in this setting implies VIL-security on all of $\{0,1\}^*$ once type-2 padding is thrown back in. Now we use the idea from Black and Rogaway [38]. In proving ALG3 secure one would be working in the information-theoretic model until the very end, so mentally replace each E_K by some first permutation π and mentally replace $D_{K'}$ by some second permutation π'. Then the $E_K \circ D_{K'} \circ E_K$ encipherment for the final block amounts to a $\pi \circ \pi' \circ \pi$ encipherment, and so, when π and π' are uniformly random permutations, this composed permutation is just another random permutation, independent of the other permutations used. That is true whether the final permutation is realized as $\pi \circ \pi' \circ \pi$, as in MAC Algorithm 3, or $\pi' \circ \pi$, as in MAC Algorithm 2; once in the information-theoretic setting, MAC Algorithms 2 and 3 are the *same*. So the security of MAC Algorithm 3 follows from the security of MAC Algorithm 2, both enjoying the same bounds.

MAC Algorithm 3 with type-1 padding inherits FIL-security from the raw CBC-MAC: the application of $E_K \circ D_{K'}$ to the CBCMAC does no harm, in the information-theoretic setting of the analysis, since the composed permutation will be independent of E_K. Similarly, it inherits



30 **algorithm** ALG3$_{K\,K'}(M)$ *MAC Algorithm 3*
31 $TAG \leftarrow \text{CBCMAC}_K(M)$
32 $Tag \leftarrow E_K(D_{K'}(TAG))$
33 $T \leftarrow \text{MSB}_\tau(Tag)$
32 **return** T

Figure 7.5: **MAC Algorithm 3.** The method is similar to MAC Algorithm 2.2 but now triple encryption of the last block is used to break for the final block. It too can be seen as symmetry-breaking device to address VIL-attacks.

provable-security to the birthday bound with type-3 padding. Neither result is "interesting"; these padding schemes make little sense for MAC Algorithms 2 or 3.

Why was MAC Algorithm 3 included, given its similarity to MAC Algorithm 2 and its needing one extra blockcipher call? One answer is the apparently improved resistance to exhaustive key search. We have already commented that, with 2^k time (but large space) and just a single message/MAC pair, one can recover the underlying key with MAC Algorithm 2, even in its two-key persona. But this is not known to be the case for MAC Algorithm 3; MAC Algorithm 3 seems "better" with respect to exhaustive key search than MAC Algorithm 2 just as triple-encryption appears better in this regard than double-encryption [140]. Just the same, forgery, or even key recovery, are not as well protected against in MAC Algorithm 3 as one might naively think: Preneel and van Oorschot [170] describe a key-recovery attack with complexity $[3 \cdot 2^k, 2^{n/2}]$. Concretely, when MAC Algorithm 3 is instantiated with DES one can, with good probability, not only find forgeries but even recover the underlying 112-bit key with $2^{32.5}$ known message/MAC pairs and $2^{57.6}$ time.

Let us describe the Preneel and van Oorschot attack—for concreteness, with DES as the underlying blockcipher and padding scheme 2. First we collect 2^{32} message/tag pairs, say for random 127-bit messages. With probability about $1 - e^{-1/2} > 39\%$ one will have a pair of messages X, X' that give rise to the same MAC. Trying all 2^{56} keys, enciphering two blocks with each, find a key K such that the input to the final CBCMAC enciphering call is the same

for X and X'. Because the key length of DES is shorter than its block length, there will usually be a unique such key. Now, spending another 2^{56} time, recover K' using the data in hand. It too will usually be uniquely determined.

Mitchell shows that substantial truncation of the MAC Algorithm 3 harms security; he gives a key-recovery attack with complexity $[2^{k+1}, (\lceil n/\tau \rceil + 1)2^{(n+\tau)/2-1}]$ [144]. For example, use of MAC Algorithm 3 with DES and truncating to 32 bits will give an algorithm susceptible to exhaustive key search after 2^{45} queries.

Resistance to key-search attacks lies outside of the conventional provable-security model. Still, one could potentially prove upper and lower bounds on $\mathbf{Adv}^{\mathrm{prf}}_{\Pi[k,n]}(t, q)$, the maximal advantage of an (information-theoretic) adversary attacking construction Π based on an ideal blockcipher with key-length k and blocklength n that makes t E-or-E^{-1} queries and q MAC queries.

Beyond the reasons given above for including both MAC Algorithms 2 and 3 of ISO 9797-1 was apparently trying to be inclusive in its choice of modes, maintaining backward compatibility with ISO 9797 and ANSI banking standards. Switching from single DES-encryption to triple DES-encryption for the final block would, in the thinking of 1980's, have been as natural a symmetry-breaking device as adding an additional encipherment.

7.14. MAC Algorithm 4. This next mode of operation is shown in Figure 7.6. The method was designed by Knudsen and Preneel for use when the underlying blockcipher is DES [114]. In that setting, the algorithm goes by the name MacDES.

MAC Algorithm 4 is meant to get around the $[3 \cdot 2^k, 2^{n/2}]$ key-recovery attacks of MAC Algorithm 3—and the more efficient key-recovery attacks still for MAC Algorithms 1 and 2. The intent is to be able to use DES in settings where 2^{56} computation is not out of the question. Of course one could say use one of MAC Algorithms 1–3 on triple-DES instead of single-DES, but MAC Algorithm 4 is more efficient—it aims to buy enhanced key length with two blockcipher calls more than the raw CBC-MAC.

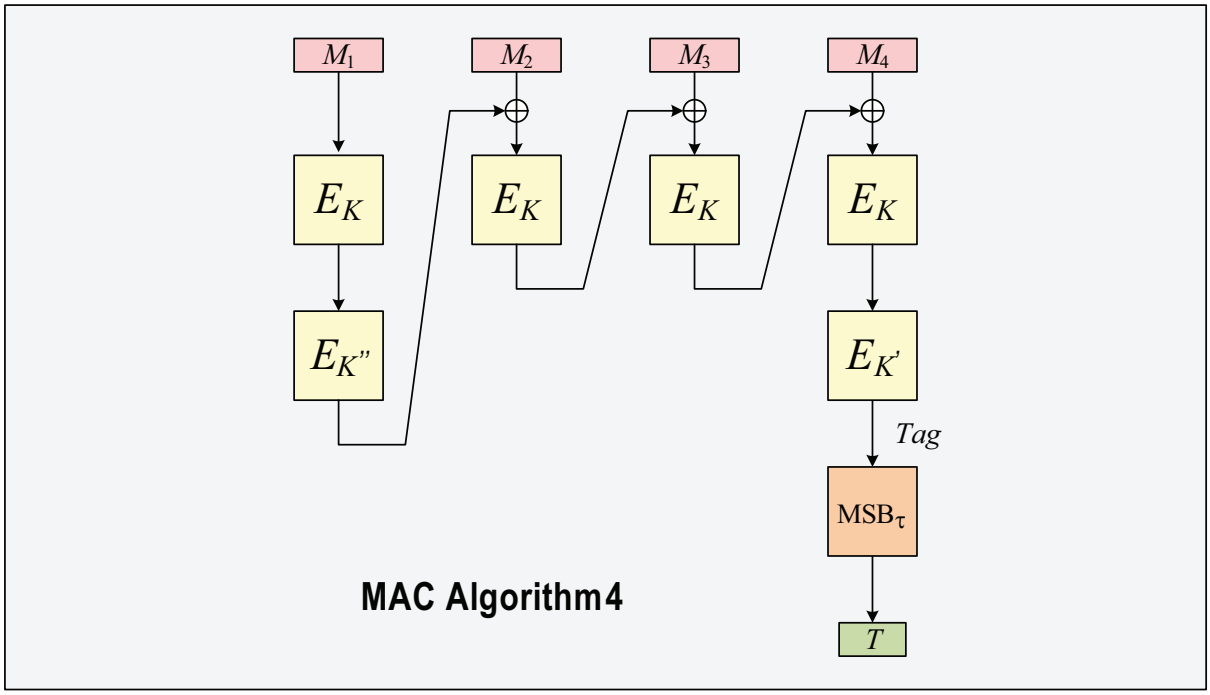
Even before ISO 9796-1 was published, Coppersmith and Mitchell described attacks on its MAC Algorithm 4 [55] when using either type-1 or type-2 padding. Let us describe the first of these attacks, assuming type-1 padding. We further assume that there is no truncation—that $\tau = n$. The attack begins by having the adversary ask arbitrary messages of at least three blocks: say the adversary asks random message of three blocks each. It does this until it finds a pair of messages $a \parallel b \parallel c$ and $d \parallel e \parallel f$ that yield the same tag T. By the birthday phenomenon, this will take about $2^{n/2}$ queries. Now the adversary strips off the final block of each of the two message, asking for the corresponding MACs; it learns the tag T_{ab} for $a \parallel b$ and it learns the tag T_{bc} for $d \parallel e$. Now observe that that, from the first part of the attack,

$$E_K(x \oplus c) = E_K(y \oplus f)$$

where x is the output of the blockcipher call that processed b (when MACing $a \parallel b \parallel c$) and, similarly, string y is the output of the blockcipher call that processed e when MACing $d \parallel e \parallel f$. From the second part of the attack, $x = D_{K'}(T_{ab})$ and $y = D_{K'}(T_{bc})$, and so

$$E_K(D_{K'}(T_{ab}) \oplus c) = E_K(D_{K'}(T_{cd}) \oplus f).$$

This means that we has a simple equation that lets one verify a correct guess of K'. The adversary therefore tries all keys K' until it finds a candidate. For a case like DES, where the key length k is less than the blocklength n, a candidate will usually be correct. Having now found K' it is easy to follow up with 2^k additional work to recover K; and then another 2^k work




```
40  algorithm ALG4_{K K'}(M)                                   MAC Algorithm 4
41  K'' ← Sep(K ‖ K')
42  M ← Pad(M)
43  if |M| < 2n then return Invalid
44  C_1 ← E_{K''}(E_K(M_1))
45  for i ← 2 to m do C_i ← E_K(M_i ⊕ C_{i−1})
46  Tag ← E_{K'}(C_m)
47  T ← MSB_τ(Tag)
48  return T
```

Figure 7.6: **MAC Algorithm 4.** The method is like MAC Algorithm 2 but is intended to enhance the effective key length of the underlying blockcipher. In fact, it does not.

to recover K''. All in all, time complexity is about $4 \cdot 2^k$ and the number of known message/tag pairs is about $2^{n/2}$.

Coppersmith and Mitchell go on to give a second attack that assumes a MAC verification oracle that is queries 2^k times, but needs only two message/tag pairs.

The authors suggest that "probably the most effective" countermeasure is to use type-3 padding instead of type-1 or type-2 padding. But, within a year, the same authors, joined now by Knudsen, had broken this scheme, too [54]. The second attack described in their paper recovers keys (K, K', K'') in time of $3 \cdot 2^k$ and using a number of chosen message/tag pairs of around $3 \cdot 2^{0.75n}$. We omit a description of the rather complex attack. Forgery attacks are more efficient still, requiring as few as one chosen message/tag pair and 2^n verification queries.

Using the bracket-notation to describe attack efficiency and assuming the underlying blockcipher is DES, using the attacks already referenced, we have a a $[0, 2^{32}]$ forgery attack and a $[2^{59}, 2^{50.6}]$ key-recovery attack when $\tau = 64$, and a $[0, 2^{33.6}]$ forgery attack and a $[2^{64}, 2^{63}]$ key-recovery attack when $\tau = 32$. (All of these bounds are adapted from Preneel [169].) These attacks are not encouraging While they do not establish that there is *no* added value to the extra blockcipher calls of MAC Algorithm 4 (compared to MAC Algorithms 2 and 3), they do

make clear that they buy a disappointing amount of added security—far less than tolerating $[2^{2k-\epsilon}, 2^{n/2-\epsilon}]$ attacks that one might, in principle, hope for in an iterated MAC that is designed to enhance the effective key length of the underlying idealized cipher.

An alternative approach to trying to obtain improved resistance to key-guessing attacks would be to use the DESX construction [110] as the underlying blockcipher in the context of, say, MAC Algorithm 2. Recall that the DESX construction takes a blockcipher $E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ and maps it to a blockcipher $E'\colon \{0,1\}^{k+n} \times \{0,1\}^n \to \{0,1\}^n$ by way of setting $E'_{K\,K'}(X) = K' \oplus E_K(X \oplus K')$. If $E' = \mathrm{DESX}[E]$ is used as the blockcipher for MAC Algorithm 2 the additional XORs cancel except for at the beginning and end. We have not investigated how the security of this construction compares to MAC Algorithm 4.

In the end, there appear to be no known results establishing improved effective key length for any CBCMAC-like construction. Are such results possible? We have no reason to think that they are not; it is probably just "accident" that such results have never been worked out. Such schemes would normally be in the ideal-cipher model. Cryptographers may have failed to notice that even when a result has a standard-model provable-security result there can *still* be good purpose to re-establishing the result in the ideal-cipher model. At issue is observing how the advantage degrades also as a function of the key length k. Were there tight formulas, in the ideal-cipher model, for both MAC Algorithm 2 and MAC Algorithm 4 one would be able to see to what extent the added initial blockcipher call was of value.

Continuing our critique of MAC Algorithm 4, we note that key separation in the mode, and even the definition of what keys are supposed to be, is quite poorly done. The spec asserts [91, page 6, column 1] that K and K' are chosen independently—the underlying key is therefore $K \parallel K'$—and that they are never to coincide. The spec also asserts that key K'' is to be derived from K' yet different from both K and K''. Achieving this set of requirements is of course impossible: if K and K' are independent k-bit strings than obviously they *can* coincide, and if K'' is derived from K' then obviously it *might* coincide with the (independent) key K randomly chosen from the same set. The most natural explanation for these conflicting demands is to imagine that the intended meaning of "different" is not the literal difference of bits strings, but something more formalistic and symbolic—the informal English-language use of a phrase like "we take three different keys" meaning that, for example, one isn't defined as being the other.

When the spec gives an example of what the authors have in mind for key separation, the example is to define K'' by complementing every other run of four bits in K'. The use of such a key-separation method would extinguish any attempt to obtain provable security under standard assumptions. A second suggestion is given that K' and K'' are to be derived from an underlying master key, which is more consistent with provable-security requirements. But using this approach would mean that the underlying key is not $K \parallel K'$, contradicting the sentence spanning pp. 5–6 of the spec. The same issues were seen in MAC Algorithm 2.

We comment that MAC Algorithm 4 does at least enjoy about the same provable security guarantees MAC Algorithm 2; the extra blockcipher call doesn't make things worse—at least if one ignores the key-separation problems and pretends all keys as distinct. This is because the key K'' could be made public, the first and second blockcipher conceptually reversed (which amounts to the same in the information-theoretic setting), and then one is simply permuting the first block by a public map. MAC Algorithm 4 fails to earn a "check" in Figure 7.1 because the semantics of a check is that one has shown the "intended" security result, not simply *some* security result, and here we regard enhanced key-length as a central part of the algorithm's goal.

We point out that there the convention of using keys K and then K'' for the first block is uglier than the opposite convention, which would have allowed MAC Algorithm 4 to coincide with MAC Algorithm 2 with an additional blockcipher call as preprocessing. It would also have facilitated the better use of CBCMAC-enabled hardware or library calls, as the algorithm would more precisely consist of the CBCMAC "with additions." The pseudocode of Figure 7.6 would have been simplified had this better abstraction boundary been used. Security would have been equivalent; we have already argued that, in the information-theoretic setting, the two modes coincide.

Our overall conclusion is that attempts to enhance key-length by the mechanism of MAC Algorithm 4 remain poorly studied, with no provable security results, and of unclear merit. Key-separation is handled with considerable clumsiness. Combined with the decline in use of DES, there is no reason to support MAC Algorithm 4 in a modern standard. Somewhat more generally, we find that the state-of-the-art in provable-security for MACs is effectively unable to distinguish the "correct" MAC Algorithm 1 with padding scheme 3, MAC Algorithm 2 with padding schemes 2 or 3, MAC Algorithm 3 with padding schemes 2 or 3, and MAC Algorithm 4 with padding schemes 2 or 3. While the history of attacks suggests that there *are* significant differences here, there are no results refined enough to distinguish among these seven schemes.

7.15. MAC Algorithm 5. The next MAC algorithm represents an interesting attempt to get beyond the birthday-attack seen in CBCMAC-like schemes. See Figure 7.7. Two "independent" repetitions of MAC Algorithm 1 are used and the results are xor'ed together.

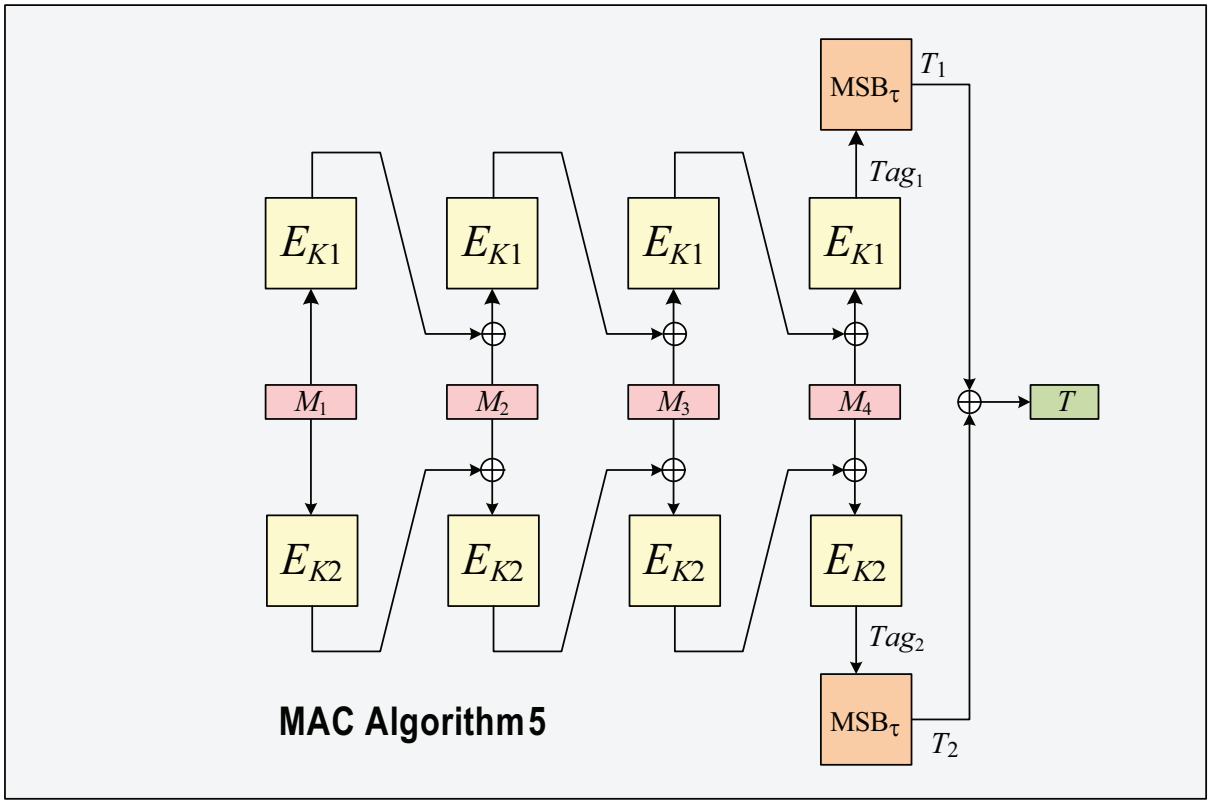
Recall that Preneel [171] and Preneel and van Oorschot [172] explain that *any* iterated cipher with an n-bit "pipe" will have forgery attacks, at least, of $2^{n/2}$ queries. All of the MAC Algorithms 1–4 are of this sort. MAC Algorithm 5 breaks out of this mold by using a $2n$-bit pipe.

Unfortunately, not only are desirable provable-security bounds absent for MAC Algorithm 5, but damaging attacks are known. For padding schemes 1 and 2, Joux, Poupard, and Stern provide a forgery attack that uses $2^{1+n/2}$ queries [105]. The hope would have been to require retain security to $2^{n-\epsilon}$ queries. Key recovery has similar complexity to this, followed a single exhaustive key-search for the underlying blockcipher.

Let us sketch how a forgery attack works, which is quite cute. Start by searching for one-block messages M and N that yield the same tag $T = E_{K1}(M) \oplus E_{K2}(M) = E_{K1}(N) \oplus E_{K2}(N)$. This requires about $2^{n/2}$ queries, assuming n-bit tags. Next compute MACs T_A for messages of the form $M \parallel A$ and MACs T_B for messages of the form $N \parallel B$ for random A, B. If it so happens that $A \oplus B = E_{K1}(M) \oplus E_{K1}(N) = E_{K2}(M) \oplus E_{K2}(N)$ then we will get collisions in the upper and lower chains, and $T_A = T_B$. When we detect the later equality, we can check if its cause was indeed collisions in the upper and lower chains and, if so, it is easy to forge. The total complexity is about $[0, 2^{n/2+1}]$.

This is a rather spectacular failure. It works for type-1 or type-2 padding. To the best of our knowledge, a good attack for type-3 padding remains open.

Were the keys $K1$ and $K2$ actually independent it would be easy to see that one has at least FIL security to the birthday-bound for with padding scheme 1, and at least VIL security to the birthday bound for padding scheme 3. Both would follow by "giving away" key $K2$, say. On the other hand, it is not obvious how to make any sort of VIL provable-security claim, even just to the birthday bound, for MAC Algorithm 5 and padding scheme 2. This seems to be an "error" in design; a mechanism aiming for beyond-birthday-bound security ought to transparently achieve at least birthday-bound security. The problem could have been remedied



MAC Algorithm 6

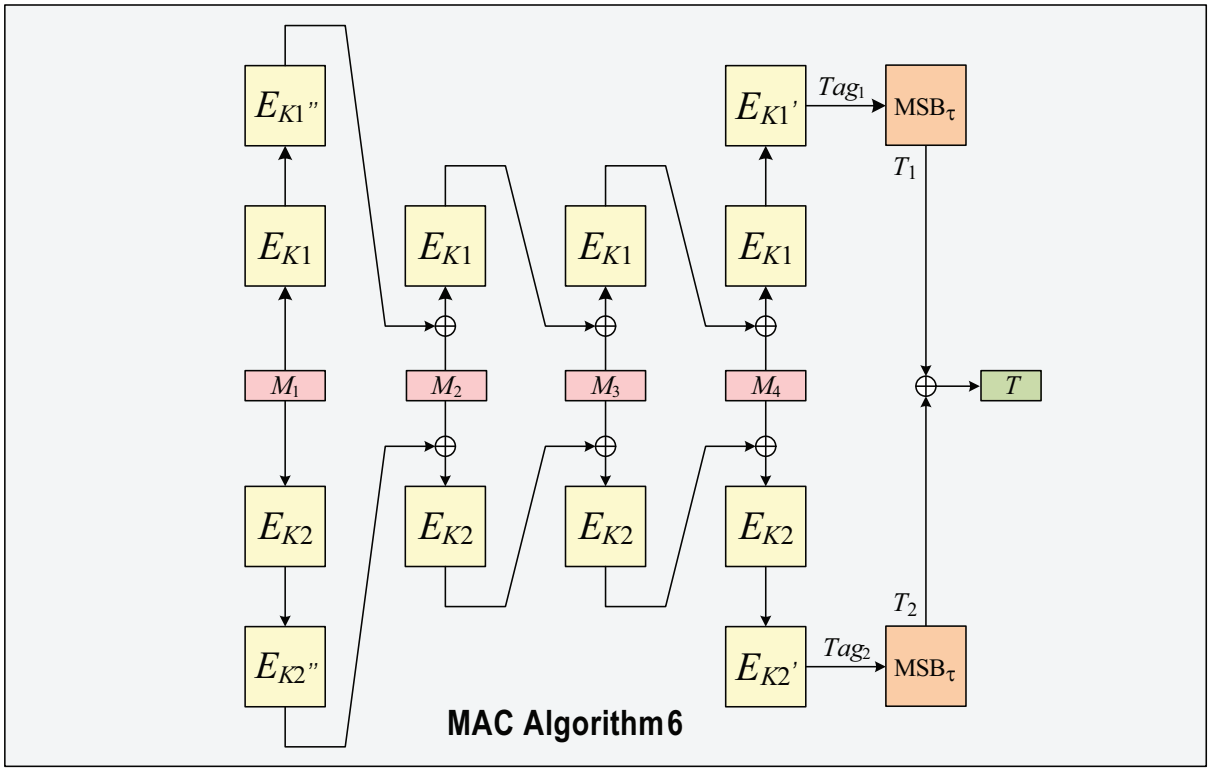
50 **algorithm** $\text{ALG5}_K(M)$
51 $K1 \parallel K2 \leftarrow \text{Sep}(K)$
52 $M \leftarrow \text{Pad}(M)$
54 $T_1 \leftarrow \text{ALG1}_{K1}(M)$
55 $T_2 \leftarrow \text{ALG1}_{K2}(M)$
56 $T \leftarrow T_1 \oplus T_2$
57 **return** T

Figure 7.7: **MAC Algorithm 5.** The algorithm consists of two "independent" executions of MAC Algorithm 1. The aim is improve security beyond the birthday bound—more than $2^{n/2}$ queries should be needed to find a forgery. The attempt fails.

by using two independent copies of MAC Algorithms 2 or 3, either of which would have made a more sensible design choice if one is trying to at least "heuristically" augment the birthday-bound security of those schemes. As we will see in a bit, the choice would have been wise.

There are key-separation problems for MAC Algorithm 5 that exactly parallel those earlier problems with key-separation in MAC Algorithms 2 and 4.

7.16. MAC Algorithm 6. The final ISO 9797-1 is like MAC Algorithm 5 but, instead of xoring two copies of MAC Algorithm 1, two copies of MAC Algorithm 4 are xor'ed together. We once again have the usual key-separation issues, here made all the more acute since three key-separation schemes must be employed: the one we have named SEP at line 61 of Figure 7.8, and the two that are implicitly parameters in the our calls to ALG4 at lines 64 and 65. Basic questions like whether the last two algorithms are expected to be the same are not answered. Note 2 of [91, Section 7.6] recommends verifying that the six keys $K1, K1', K2', K2, K2', K2''$ are all distinct—a constraint that cannot be guaranteed for any algorithms SEP and Sep, and contrary to the concrete recommendation in [91, Note 1, Section 7.6]. Once again, key-



60 **algorithm** ALG6$_{K \,\|\, K'}(M)$ *MAC Algorithm 6*
61 $K1 \,\|\, K1' \,\|\, K2 \,\|\, K2' \leftarrow \text{SEP}(K \,\|\, K')$
62 $M \leftarrow \text{Pad}(M)$
64 $T_1 \leftarrow \text{ALG4}_{K1\,K1'}(M)$
65 $T_2 \leftarrow \text{ALG4}_{K2\,K2'}(M)$
66 $T \leftarrow T_1 \oplus T_2$
67 **return** T

Figure 7.8: **MAC Algorithm 6.**

separation seems to have been added as a careless afterthought. We prefer to think of the six keys as random and independent, although such a choice would seem to be explicitly forbidden by the language of the standard.

Pretending the keys to be independent, MAC Algorithm 6 would at least enjoy provable-security to the birthday bound, with any of the three padding regimes: give away the key $K2$, say, and we are back to Algorithm 4, which, for all its problems, does at least enjoy provable security to the birthday bound.

Quite recently—and rather unexpectedly—Kan Yasuda proved that MAC Algorithm 6 *does* achieve provable-security beyond the birthday bound [206]. The lovely result builds on work of Lucks and others [129] about creating an n-bit PRF by xoring together two n-bit PRPs. Let Π = ALG6 with type-2 padding but using six random and independent n-bit permutations. Let A be an adversary that asks at most q queries, each having $m - 1$ or fewer blocks. Then Yasuda shows [206, Theorem 1] that

$$\mathbf{Adv}^{\text{prf}}_{\pi}(A) \leq \frac{12m^4q^3}{2^{2n}} \tag{7.1}$$

and, if $m \leq 2^{2n/5}$, then

$$\mathbf{Adv}^{\mathrm{prf}}_{\pi}(A) \leq \frac{40m^3q^3}{2^{2n}} \; . \tag{7.2}$$

While the bounds are somewhat disappointing when one sticks in DES parameters (the $2^{2n/5}$ cutoff is only $2^{25.6}$ when $n = 64$) the work is the first to show a beyond-birthday-bound result for any (deterministic, stateless) CBCMAC-like scheme.

While encouraging, the new result does not suggest that ALG6 itself is a particularly good choice. First, there is the problem of key-separation, which should be completely redone, either demanding all independent keys or providing an explicit and provable-security-friendly mechanism. Second, the key-length extension idea of using the initial extra blockcipher call was essentially discredited by Coppersmith, Knudsen, and Mitchell [54, 55] in the context of MAC Algorithm 4, so one should be skeptical that it will work in the context of MAC Algorithm 6. More work is needed to give any significant evidence that the MAC achieves both of its stated goals—beyond-birthday-bound security *and* increased effective key length.

7.17. Concluding remarks. The known attacks, the conjectured properties, and the provable-security results associated to the ISO/IEC 9797-1:1999 algorithms comprise a complicated patchwork. It is, frankly, a bit of a mess. Brincat and Mitchell interpret the sate of affairs by saying that

> it is important for users to carefully assess the significance of the various MAC attacks in the context of the environment in which the resulting MAC algorithm is to be used. There may be no benefit from using certain sophisticated MAC systems in an environment which has other security features in operation which make attacks against simpler MAC schemes impossible to carry out. [47, Section 7].

We would draw essentially the opposite conclusion. It is just too difficult for even a sophisticated user to properly navigate the myriad of algorithms of ISO 9797-1, understanding what is known about each and what would make for a sensible algorithmic choice for some particular application. For multiple algorithms and intended properties, nobody has provided any proof-based guarantees evidencing that the algorithm has the intended property. Many of the algorithms have been found to have significant weaknesses, the discoveries sometimes happening long after the algorithms were first put forward. The underspecification of key-separation makes it impossible to retain standard-model provable security claims, or even to produce good test vectors. Taken together, these criticisms suggest, to me, that there is something rather wrong with the standard. It admitted too many techniques, included deficient ones, and left things overly open-ended.

Chapter 8

CMAC Mode

8.1. Summary. This chapter looks at the CMAC mode (Cipher-based MAC) of NIST Recommendation SP 800-38B [63].

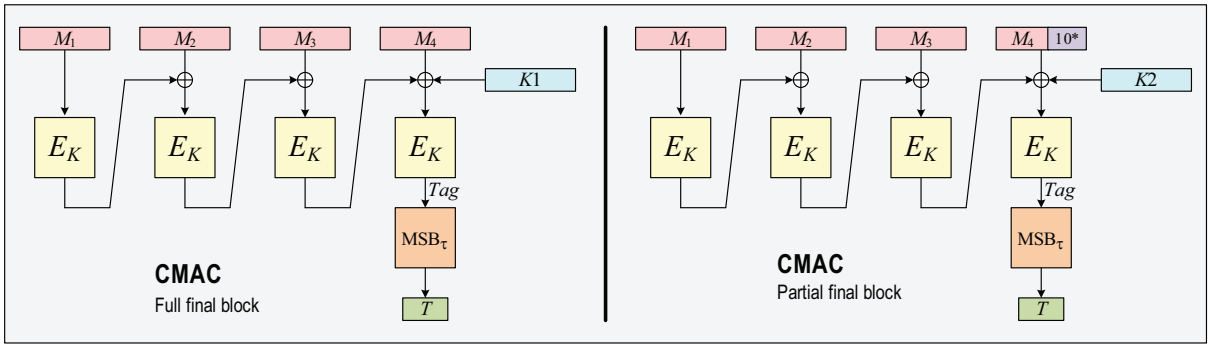
CMAC is a simple and clean variant of the CBC-MAC. Used with a strong 128-bit blockcipher, the scheme enjoys good provable-security properties—reasonable bounds under standard assumptions—with no significant identified flaws. In recent years its provable security bounds have improved, furthering our understanding of this mode. Especially when set against the morass of ISO/IEC 9797-1:1999, NIST's document seems clear, concise, and informative.

The biggest security issue with CMAC is that relatively frequent key changes are advisable if one uses a 64-bit blockcipher. This is true for most blockcipher modes of operation. Also, the mode is not a desirable choice to use with DES, as key-search attacks will then be computationally easy.

There are some efficiency issues with CMAC, issues shared with any CBCMAC-based scheme: the inherently serial nature of CMAC will limit its performance on blockcipher-enabled hardware, and software speeds will be limited to that which is achievable by the underlying blockcipher. Mechanism misuse is certainly possible, particularly if, overlooking the spec's requirements, a user uses $E_K(0^n)$ for some purpose other than deriving subkeys $K1$ and $K2$. According to Preneel [169], this particular value has been used by banks as a key-confirmation token. But this is not really CMAC's "fault," and NIST's Recommendation is careful to warn against this and other usage errors.

8.2. Definition of the scheme. The CMAC mode of operation is defined and illustrated in Figure 8.1. CMAC depends on two parameters: a blockcipher $E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ and a tag length $\tau \in [1..n]$. Fixing these parameters, CMAC maps a key $K \in \{0,1\}^k$ and a message $M \in \{0,1\}^*$ to a τ-bit tag $T = \mathrm{CMAC}_K(M)$. Assuming subkeys $K1$ and $K2$ have been precomputed (their computation takes one blockcipher call), CMAC authenticates message M using $\max\{1, \lceil |M|/n \rceil\}$ blockcipher calls. This is better than all of the ISO 9797-1 schemes, and is in some sense optimal for a CBCMAC-like scheme. According to the spec, the blockcipher underlying CMAC must be NIST *approved*, which limits the choices to AES, TDEA (triple-DES), and Skipjack.

The definition we give for CMAC depends on a function $\mathrm{dbl}(\cdot)$ that maps n-bit strings to n-bit strings. The routine realizes a multiplication in $\mathrm{GF}(2^n)$ of its input $X \in \{0,1\}^n$ by the constant $u = \mathbf{2} = 0^{n-1}10$. A fixed irreducible polynomial $g(u)$ is used to represent field points. The selected polynomial is the lexicographically first one among all irreducible polynomials of degree n with a minimum number of nonzero terms. Here, lexicographically-first entails writing



10 **algorithm** $\mathrm{CMAC}_K(M)$ *CMAC mode*
11 $K1 \leftarrow \mathrm{dbl}(E_K(0^n))$
12 $K2 \leftarrow \mathrm{dbl}(K1)$
13 **if** $|M| \in \{n, 2n, 3n, \ldots\}$
14 **then** $K' \leftarrow K1$, $P \leftarrow M$
15 **else** $K' \leftarrow K2$, $P \leftarrow M10^i$ where $i = n - 1 - (|M| \bmod n)$
16 $M_1 \cdots M_m \leftarrow M$ where $|M_1| = \cdots = |M_m| = n$
17 **for** $i \leftarrow 1$ **to** m **do** $C_i \leftarrow E_K(M_i \oplus C_{i-1})$
18 $T \leftarrow \mathrm{MSB}_\tau(C_m)$
19 **return** T

Figure 8.1: **CMAC Mode.** The scheme is parameterized by a blockcipher $E\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$s and a tag length $\tau \in [1..n]$. **Left:** the case where the message length is a positive multiple of n bits. **Right:** the case when it is not. The values $K1$ and $K2$ are derived from K by "doubling" once or twice, where, here, to "double" means to multiply by the point in $\mathrm{GF}(()2^n)$ whose representation is $u = 0^{n-1}10$.

of the polynomial, omitting its high-order term, from degree $n-1$ term (leftmost) on down. Concretely, for $n = 128$ we use the irreducible polynomial $g(u) = u^{128} + u^7 + u^2 + u + 1$ while for $n = 64$ we use $g(u) = u^{64} + u^4 + u^3 + u + 1$. There are no NIST-approved blockciphers for any other blocklength. Using these two polynomial, we have that

$$\mathrm{dbl}(X) \quad = \quad \begin{cases} X \ll 1 & \text{if } \mathrm{MSB}_1(X) = 0, \text{ and} \\ X \ll 1 \oplus 0^{120}10000111 & \text{if } \mathrm{MSB}_1(X) = 1 \end{cases} \tag{8.1}$$

when $X \in \{0,1\}^{128}$, while

$$\mathrm{dbl}(X) \quad = \quad \begin{cases} X \ll 1 & \text{if } \mathrm{MSB}_1(X) = 0, \text{ and} \\ X \ll 1 \oplus 0^{59}11011 & \text{if } \mathrm{MSB}_1(X) = 1 \end{cases} \tag{8.2}$$

when $X \in \{0,1\}^{64}$. Here $X \ll 1$ is the left-shift of X by one bit position, the bit 0 entering in at the right.

The specification document speaks of MAC generation and MAC verification as separate algorithms, but remember that the way we have defined MACs there is only one function to deal with: verification consists of recomputing the anticipated MAC and comparing. The specification also speaks of a separate subkey-generation procedure, but we have folded this into our definition for the scheme.

8.3. History. There is a clear evolution of ideas leading to the CMAC algorithm. We sketch it as happening in four steps.

8.3.1 **Basic CBC-MAC**. We begin with the basic CBC-MAC of ANSI X9.9 [4], which has two fundamental problems: (a) it is only defined on strings that are a multiple of n

bits; and (b) it does not achieve VIL security, only FIL security. Both issues were discussed in §7.8 of this report. The same issues motivate many of the ISO 9797-1 MAC Algorithms 1–6.

8.3.2 **EMAC**. We fix the problems just mentioned by evolving to MAC Algorithm 2, padding method 2, of ISO 9797-1. The scheme adds obligatory 10*-padding, applies the raw CBC-MAC, then enciphers the result, using the same blockcipher with a new key. As described in Chapter 7, the algorithm was first described under the RACE project [45], and was first proven secure, in the provable-security tradition, assuming the underlying blockcipher is a PRP, by Petrank and Rackoff [167].

8.3.3 **XCBC**. Next we trim the extra blockcipher calls by using a little trick. First, instead of obligatory 10*-padding we refrain from using any padding at all if the message is already a positive multiple of n-bits. We continue to use 10*-padding otherwise. Second, instead of enciphering the last block with the permutation $E_{K'}(E_K(\cdot))$ we encipher the final block with the permutation $E_K(K1 \oplus \cdot)$ for the case where we used no padding, and $E_K(K2 \oplus \cdot)$ for the case where we used 10* padding. The underlying MAC key is now $K \parallel K1 \parallel K2 \in \{0,1\}^{k+2n}$. This "three-key construction" was invented by Black and Rogaway and named XCBC [38, 39]. We proved the mode secure, up to the birthday bound, if the underlying blockcipher is a PRP.

8.3.4 **OMAC**. Finally, to minimize the length of the needed key, the cost of key-setup, and to make sure one is not misrepresenting the algorithm's strength with a misleadingly long key, we define keys $K1$ and $K2$ from the single underlying key K by using a bit of finite-field arithmetic. This step was designed by Iwata and Kurosawa [96] (following a prior reduction from three keys to two, TMAC [119]). After implementing a minor tweak suggested by Rogaway [98], the algorithm was slightly revised by the authors and renamed OMAC1. NIST's CMAC mode *is* OMAC1, apart from its details like what blockciphers may be used.

Subsequent standardization of CMAC includes RFC 4493 and RFC 4494 [99, 100].

8.4. Provable-security results. We consider the information-theoretic setting where the underlying keyed blockcipher E_K for CMAC is replaced by a uniformly random permutation π. An adversary $\mathcal{A}$, then, is trying to distinguish $\mathrm{CMAC}_\pi(\cdot)$, for a random permutation π on n bits, from a random function $\rho(\cdot)$ from $\{0,1\}^*$ to $\{0,1\}^n$. One aims for a formula f that tightly upper-bounds the distinguishing advantage that the adversary can get in terms of the resources of the adversary may use—resources that include the total number q of oracle queries the adversary makes; the total block length σ for all of those queries (round up the bit length to the next positive multiple of n, then divide by n); and the maximal block length m of any query (use the same rounding as before).

The initial bounds for CMAC's security, by Iwata and Kurosawa [96, Lemma 5.2], took the form blockcipher,

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathrm{CMAC}}(m,q,\sigma) \;\leq\; \frac{5m^2q^2}{2^n}\,. \tag{8.3}$$

Here and subsequently we ignore lower order terms (eg, the "actual" formula in the paper has $(5m^2+1)q^2$ in the numerator). In follow-on work, the same authors slightly improve and recast their bounds in terms of σ instead of m and q [97], obtaining

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathrm{CMAC}}(m,q,\sigma) \;\leq\; \frac{4\sigma^2}{2^n}. \tag{8.4}$$

Since $\sigma \leq mq$, the new bound is always better, and the difference can be substantial if message length substantially vary.

The best provable-security bounds on OMAC to date are due to Nandi [147], who shows that

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathrm{CMAC}}(m, q, \sigma) \quad \leq \quad \frac{5\sigma q}{2^n}. \tag{8.5}$$

The improvement is to trade a σ for a $q \leq \sigma$. Recent work by the same author generalizes his proof technique [149].

The bounds above are all for the information-theoretic setting. For the complexity-theoretic setting one just adds in an extra term measuring the PRP-insecurity of the underlying blockcipher.

Let us see what the quantitative bounds mean in practice, using the last formula as our guide. When the underlying blockcipher has a 128-bit blocksize, say AES, Appendix B of SP 800-38B recommends using CMAC for no more than 2^{48} messages. If each of 2^{48} messages is 1 MB in length, we would know that the maximal adversarial advantage in distinguishing CMAC from a random function would be at most $5 \cdot (10^9/16) \cdot 2^{48} \cdot 2^{-128} < 2^{-51}$ plus the insecurity of the underlying blockcipher, a very strong guarantee. The same appendix suggests that, for a 64-bit blockcipher, say TDEA, one change keys after 2^{21} messages. If this recommendation is followed and messages are again 1 GB or less, the maximal advantage will be bounded by $5 \cdot (10^9/16) \cdot 2^{21} \cdot 2^{-128} < 2^{-14}$ plus the insecurity of the blockcipher, still an excellent guarantee. Of course all bets are off if the additive term we have ignored quantifying the PRP-insecurity of AES or TDEA as a PRP is not insignificant, but the likelihood of this is extremely low.

8.5. Attacks. The birthday-bound attack on CMAC is tight in the sense that any iterated hash function with an n-bit pipe will have forgery attacks that involve asking about $q = 2^{n/2}$ queries, each of short length. The attack is often credited to Preneel and van Oorschot [172]; see §7.11 for a description. The gap, then is only insofar as the best existing security bounds that drop off in $\sigma q/2^n$ while the natural forgery attack gets advantage $q^2/2^n$. Note that the birthday attacks can be more damaging than "just" identifying a forgery; particular forgeries can be targeted, as described by Jia, Wang, Yuan, and Xu [102].

In public comments to NIST, Mitchell points to the birthday attack on CMAC [145], describing it in a couple of forms, and he describes key-recovery attacks on the mode as well. He regards OMAC as weaker than MAC Algorithm 2 (§7.12) and concludes that the former mode should not be adopted by NIST. But Mitchell's conclusions are not supported by his accompanying analysis, which only demonstrates rather standard attacks, of comparable complexity, on CMAC, EMAC, and other modes. As Iwata reasonably responds [94], he and his coauthor "did not show [attacks] achieving advantage $\Omega(\sigma^2/2^n)$ because every standard mode of operation (including XCBC, TMAC, OMAC, EMAC, ...) has been susceptible to attacks of this (in)security. It is what everyone expects."

As with MAC Algorithm 2, key-guessing attacks on CMAC will be effective if the underlying blockcipher admits reasonable key-guessing attacks; in 2^k time, using a couple of known plaintext/MAC pairs, the underlying key can be recovered. This should not be considered a damaging attack; the mode was not designed to increase the effective key length of the underlying blockcipher.

8.6. Minor comments on the spec itself. As indicated earlier, NIST Recommendation 800-38B is a well executed document. Perhaps the worst complaint we might make is to note

that the document ascribes two different meanings to the term *message span* [63, Appendix B]. First we are told that this is the *number* of messages MACed and should be limited to 2^{48} when $n = 128$, and limited to 2^{21} when $n = 64$. Then we are told that the message span may instead be understood as the number of blocks of that will get MACed—"[f]or applications where higher confidence in the security is required." This sends mixed signals. Note that the latter notion of message span more closely comports with the provable-security results; in principle, MACing a single extremely long message could be insecure. In fact, as with other CBCMAC-variants, one *can* give a pair of messages M, M' that would generate the same tag under CMAC—the problem is that the two messages would be absurdly long.[1]

The prescribed name of the algorithm, "Cipher-based MAC" [63, p. 1], is rather vacuous. Indeed I had always assumed that the name "CMAC" was meant to suggest a CBC-based MAC, which is at least a bit more narrowly prescribed.

I might mention that subkeys $K1$ and $K2$ are regarded as "perquisites" for MAC Verification [63, p. 10] (but not for MAC Generation). They don't belong here. More generally, the notion of "prerequisites," here including the key, seems flimsy. It might be better to regard the arguments of the MAC as what is needed in a security definition, and to regard anything else as a parameter. Finally, there I would mention a typo in [63, footnote 3, p. 13]: C_m should be C_n.

Overall, NIST SP 800-38B is an elegant specification document for a simple and provably-sound mode.

[1] For example, let M consists of N zero blocks, and let M' consist of twice this number of zero blocks, where N is the product of all prime factors less than 2^n. It is not hard to see that these messages have the same tag under CMAC, namely $T = E_K(K_1)$.

Chapter 9

HMAC Mode

9.1. Summary. HMAC provides a simple way to key a cryptographic hash function to create a message authentication code. It is used both as a PRF and as a MAC. The simplicity and strength of the construction has lead to widespread standardization and use; the mechanism is not only standardized in NIST FIPS 198-1 [159] but is also specified in RFC 2104, IEEE 802.11, SSL 3.0, TLS 1.0–1.2, SSH, and S-HTTP. The mode was designed by Bellare, Canetti, and Krawczyk [13].

Somewhat confusingly, the term HMAC is used to refer to several algorithms that share a common core but differ in their keylengths and the way they derive their subkeys. We will distinguish HMAC0 [11], HMAC1 [13], and HMAC2 (FIPS 198-1 and RFC 2104), referring to the family collectively as HMAC. Each HMAC algorithm is itself derived from the NMAC algorithm of [13] via a key-derivation function (KDF).

The NMAC algorithm is supported by a proof that guarantees it is a good PRF under the assumption that the compression function underlying the hash function it uses is itself a good PRF [11]. Each HMAC algorithm inherits this security under the additional assumption that its KDF is a good PRG (pseudorandom generator).

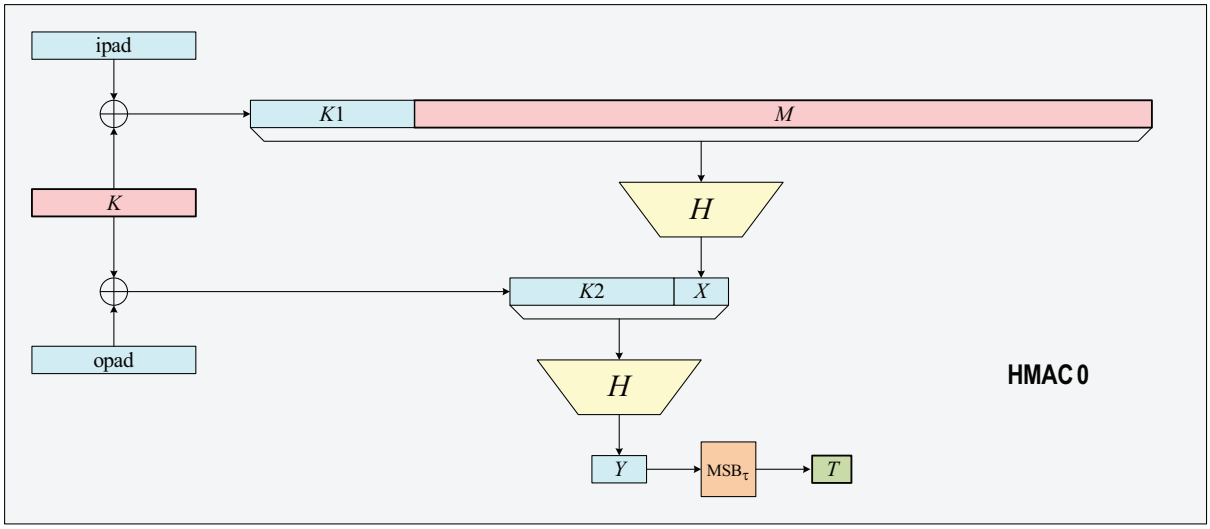
Being iterated constructions, the (untruncated) HMAC algorithms are subject to the usual birthday attack [172]. The proof-based bounds effectively assert that this is best possible attack as long as the compression function is a good PRF.

Is this last assumption true? Obviously, it depends on the hash function. Right now, we have no attacks refuting this assumption, not only for SHA1 but even for MD5.

Extensive efforts have been made to attack HMAC directly [53, 70, 111, 173, 196, 200]. They have, however, made hardly a dent; PRF attacks more effective than the birthday remain unknown for HMAC-MD5 and HMAC-SHA1. This is evidence of the pseudorandomness of the compression functions.

An intriguing part of HMAC's story has been its strength even when implemented with hash functions whose collision-resistance is compromised. Thus, collisions have been found for MD5 [36, 190, 191, 199] but this has not given rise to better-than-birthday PRF-attacks on HMAC-MD5. An explanation for this phenomenon is that finding a compression function not to be collision-resistant should in no way be construed as evidence that it is not a good PRF.

Overall, HMAC remains a sound choice for a PRF and MAC. It is easy to implement and its security has held up very well.



000 **algorithm** $\text{NMAC}_{L1 \,\|\, L2}(M)$
001 $X \leftarrow h^*_{L1}(M \,\|\, \mathsf{pad}(b + |M|)$
002 $Y \leftarrow h_{L2}(X \,\|\, \mathsf{pad}(b + c))$
003 **return** Y

000 **algorithm** $\text{HMAC0}_K(M)$
001 $K1 \leftarrow K \oplus \mathsf{ipad}$
002 $K2 \leftarrow K \oplus \mathsf{opad}$
003 $X \leftarrow H(K1 \,\|\, M)$
004 $Y \leftarrow H(K2 \,\|\, X)$
005 $T \leftarrow \text{MSB}_\tau(Y)$
006 **return** T

010 **algorithm** $\text{KDF0}(K)$
011 $L1 \leftarrow h_{IV}(K \oplus \mathsf{ipad})$
012 $L2 \leftarrow h_{IV}(K \oplus \mathsf{opad})$
013 **return** $L1 \,\|\, L2$

020 **algorithm** $\text{HMAC0}_K(M)$
021 $L1 \,\|\, L2 \leftarrow \text{KDF0}(K)$
022 $Y \leftarrow \text{NMAC}_{L1 \,\|\, L2}(M)$
023 $T \leftarrow \text{MSB}_\tau(Y)$
024 **return** T

100 **algorithm** $\text{HMAC1}_K(M)$
101 $K \leftarrow K \,\|\, 0^{b-c}$
102 $T \leftarrow \text{HMAC0}_K(M)$
103 **return** T

110 **algorithm** $\text{KDF1}(K)$
111 $K \leftarrow K \,\|\, 0^{b-c}$
112 $L1 \,\|\, L2 \leftarrow \text{KDF0}(K)$
113 **return** $L1 \,\|\, L2$

120 **algorithm** $\text{HMAC1}_K(M)$
121 $L1 \,\|\, L2 \leftarrow \text{KDF1}(K)$
122 $Y \leftarrow \text{NMAC}_{L1 \,\|\, L2}(M)$
123 $T \leftarrow \text{MSB}_\tau(Y)$
124 **return** T

200 **algorithm** $\text{HMAC2}_K(M)$
201 **if** $|K| > b$ **then** $K \leftarrow H(K)$
202 $K \leftarrow K \,\|\, 0^{b-|K|}$
203 $T \leftarrow \text{HMAC0}_K(M)$
204 **return** T

210 **algorithm** $\text{KDF2}(K)$
211 **if** $|K| > b$ **then** $K \leftarrow H(K)$
212 $K \leftarrow K \,\|\, 0^{b-|K|}$
213 $L1 \,\|\, L_2 \leftarrow \text{KDF0}(K)$
214 **return** $L1 \,\|\, L2$

220 **algorithm** $\text{HMAC2}_K(M)$
221 $L1 \,\|\, L2 \leftarrow \text{KDF2}(K)$
222 $Y \leftarrow \text{NMAC}_{L1 \,\|\, L2}(M)$
223 $T \leftarrow \text{MSB}_\tau(Y)$
224 **return** T

Figure 9.1: **HMAC mode.** The HMAC family of algorithms is parameterized by an iterated hash function $H\colon \{0,1\}^{\le d} \to \{0,1\}^c$ and a tag length $\tau \in [1..c]$. HMAC0 is the version of [11], which takes a b-bit key. HMAC1 is the original construction [13], which uses a c-bit key. HMAC2 is its extension, as in FIPS 198-1; it that takes a key of any length. In each case, we first show how the mode is defined in terms of H and then show how to define it from NMAC via a key-derivation function. NMAC takes a $2c$-bit key $L1 \,\|\, L2$ which it views as c-bit halves. Above, $h\colon \{0,1\}^c \times \{0,1\}^b \to \{0,1\}^c$ is the compression function, h^* is its iteration, pad is the padding function, and IV is the initial vector defining H. The picture illustrates HMAC0.

9.2. Iterated hash functions. A hash function is a map $H\colon \{0,1\}^{\leq d} \to \{0,1\}^c$ that takes as input a string of length at most d bits and returns a c-bit string, for values d, c associated to the function. The function is keyless and has an associated blocklength b such that $b > c$ and b is divisible by 8. Examples of hash functions are MD5 ($b = 512$, $c = 128$, $d = \infty$) and SHA1 ($b = 512$, $c = 160$, $d = 2^{64} - 1$).

The HMAC algorithm uses an iterated hash function. This means H is built by iterating an underlying compression function h as we now describe.

A compression function is a map $h\colon \{0,1\}^c \times \{0,1\}^b \to \{0,1\}^c$ that takes a chaining variable and a message block and returns an updated chaining variable. The blocklength b must be greater than c and must be a multiple of 8, and it will become what we referred to above as the blocklength of the hash function. Let $B = \{0,1\}^b$ be the set of all possible blocks and let B^+ be the set of all strings whose length is a positive multiple of b. Fix a maximum message length $d < 2^b$ and a padding function pad that takes input an integer and returns a string. The function pad must have the following properties (1) $M^* = M \parallel \mathsf{pad}(|M|) \in B^+$ for all M; (2) $M_1 \neq M_2$ implies $M_1 \parallel \mathsf{pad}(|M_1|) \neq M_2 \parallel \mathsf{pad}(|M_2|)$; and (3) $\mathsf{pad}(b + c) \in \{0,1\}^{b-c}$. For $X \in B^+$ and $L \in \{0,1\}^c$ we define

algorithm $h^*_L(X)$
$C[0] \leftarrow L$
$X[1] \ldots X[m] \leftarrow X$ where $|X[i]| = b$
for $i \leftarrow 1$ **to** m **do** $C[i] \leftarrow h(C[i-1], X[i])$
return $C[m]$

Here $X[1] \ldots X[m] \leftarrow X$ means we parse X into b-bit blocks. Finally, the hash function H is defined via $H(M) = h^*_{IV}(M^*)$ for all M of length at most d, where IV is a fixed c-bit string associated to the hash function.

Hash functions MD4, MD5, SHA1, and RIPEMD are all of this type. In these cases, $d = 2^{64} - 1$. (For the first two examples, it has been conventional to regard d as "infinite"—there is no bound on the length of the string to be hashed—and to annotate the length as $|M| \bmod 2^{64}$ rather than $|M|$.) Some SHA3 candidates are not iterated, however.

According to FIPS 198-1, the hash function must be an *approved, iterated* one. At present, there are five NIST-approved hash functions (all of them iterated): SHA-1, SHA-224, SHA-256, SHA-384, SHA-512. The last four are collectively referred to has the SHA-2 family. While US federal agencies must stop using SHA-1 for digital signatures, digital time stamping, and other applications that require collision resistance, SHA-1 may continue to be used for HMAC and various other applications.

9.3. Definition of the scheme. Figure 9.1 first defines NMAC [13], which takes a $2c$ bit key regarded as the concatenation of two c-bit keys. Note that condition (1) on pad, namely that $M \parallel \mathsf{pad}(|M|) \in B^+$ for all M, also ensures that $M \parallel \mathsf{pad}(b + |M|) \in B^+$, so that the input to h^*_{L1} at line 001 is indeed in B^+. Also condition (3) on the padding function, namely that $\mathsf{pad}(b + c) \in \{0,1\}^{b-c}$, ensures that the input to h_{IV} at line 002 of NMAC is indeed b bits long.

The figure goes on to define HMAC0 [11], which has a b-bit key; HMAC1 [13], which has a c-bit key; and HMAC2 (FIPS 198-1, RFC 2104), which has a key of any length. The strings ipad and opad are distinct b-bit constants. Specifically, ipad is the byte $\mathtt{0x36}$ repeated $b/8$ times and opad is the byte $\mathtt{0x5C}$ repeated $b/8$ times. Recall that the function $\mathrm{MSB}_\tau(Y)$ returns the first τ bits of the string Y where τ, along with the iterated hash function H, are parameters of

Mode	Keylength	Assumption under which the mode is a good PRF
NMAC	$2c$	h is a good PRF
HMAC0	b	NMAC is a good PRF and KDF0 is a good PRG
HMAC1	c	NMAC is a good PRF and KDF1 is a good PRG
HMAC2	any	NMAC is a good PRF and KDF2 is a good PRG

Figure 9.2: **Provable-security of HMAC**. The table provides a summary of known results.

the scheme.

The HMAC algorithms can be implemented directly and in a blackbox way from the hash function, and the first-column definitions present the schemes this way. The third column is an alternative and equivalent formulation of the algorithms, showing how they can be viewed as first applying a key-derivation function (KDF) to the key and then applying NMAC, the KDF being depicted in the second column. This more intrusive view is more conducive to the analysis.

Let us briefly explain why the third column formulations of the algorithms are in fact equivalent to the first column formulations. It suffices to show that $\mathrm{HMAC0}_K(M) = \mathrm{NMAC}_{L1 \,\|\, L2}(M)$ for any b-bit K, where $L1 = h_{IV}(K1)$ and $L2 = h_{IV}(K2)$. Indeed, in the computation $X \leftarrow H(K1 \,\|\, M)$, the subkey $K1$ is b-bits long and hence $X = h^*_{L1}(M \,\|\, \mathsf{pad}(b + |M|))$. Now X is c-bits long and $K2$ is b-bits long, so $Y = H(K2 \,\|\, X) = h^*_{L2}(X \,\|\, \mathsf{pad}(b+c)) = h_{L2}(X \,\|\, \mathsf{pad}(b+c))$, the last equality because $\mathsf{pad}(b + c) \in \{0,1\}^{b-c}$.

It causes some confusion that "HMAC" refers to different things in the literature. As the above illustrates, the differences relate to the key-length. See Figure 9.2.

9.4. Security proofs. HMAC is used not only as a MAC but also as a PRF (for example, for key derivation). Proofs accordingly aim to establish that HMAC is a PRF. They assume the underlying hash function H is iterated.

Before we state results, we recall that when $F\colon \{0,1\}^k \times D \to R$ is said to be a PRF, it is being keyed by its first input. In particular, when the compression function $h\colon \{0,1\}^c \times \{0,1\}^b \to \{0,1\}^c$ is assumed to be a PRF, it is keyed by its c-bit chaining variable. Also recall that $G\colon \{0,1\}^k \to \{0,1\}^s$ is a PRG if its output on input a random key is indistinguishable from a random string of length s. Formally, the prg-advantage of an adversary $\mathcal{A}$ is defined via $\Pr[\mathcal{A}(G(K)) \Rightarrow 1] - \Pr[\mathcal{A}(S) \Rightarrow 1]$ where K is chosen at random from $\{0,1\}^k$ and S is chosen at random from $\{0,1\}^s$.

The following discussion assumes there is no truncation. We discuss the effects of truncation later.

Each HMAC variant can be proven to be a PRF under two assumptions. The first is that NMAC$\colon \{0,1\}^{2c} \times \{0,1\}^{\leq d-b} \to \{0,1\}^c$ is a PRF. The second is that the KDF corresponding to this HMAC variant is a PRG. In more detail, HMAC0$\colon \{0,1\}^b \times \{0,1\}^{\leq d-b} \to \{0,1\}^c$ can be proved to be a PRF assuming NMAC is a PRF and KDF0$\colon \{0,1\}^b \to \{0,1\}^{2c}$ is a PRG. HMAC1$\colon \{0,1\}^c \times \{0,1\}^{\leq d-b} \to \{0,1\}^c$ can be proved to be a PRF assuming NMAC is a PRF and KDF1$\colon \{0,1\}^c \to \{0,1\}^{2c}$ is a PRG. For HMAC2 we must first fix some (any) keylength k. Then, HMAC2$\colon \{0,1\}^k \times \{0,1\}^{\leq d-b} \to \{0,1\}^c$ can be proved to be a PRF assuming NMAC is a PRF and KDF2$\colon \{0,1\}^k \to \{0,1\}^{2c}$ is a PRG. See Figure 9.2.

The PRG assumptions on the KDFs are relatively mild since there are no adversary-chosen

inputs. The core, security-wise, is thus NMAC. Let us now talk of its security.

Letting $h\colon \{0,1\}^c \times \{0,1\}^b \to \{0,1\}^c$ denote the compression function underlying the hash function H, Bellare, Canetti and Krawczyk (BCK) [13] prove that NMAC is a secure PRF under two assumptions. The first assumption is that h is a PRF. The second assumption is that H is collision resistant (CR).

At the time BCK was written, in 1996, the assumption that H was CR seemed eminently reasonable; collision resistance was the design goal of hash functions. But, in 2004, collisions were found for MD5 [199]. Now MD5 is considered broken from the point of view of collision resistance [36, 190, 191, 199], and SHA1 is considered weak [198].

Interestingly, the collision-finding attacks on MD5 and SHA1 did not give rise to any PRF or forgery attacks on HMAC or NMAC with either of these hash functions. However, the BCK result [13] was rendered void since one of its assumptions had been falsified. In 2006, Bellare [11] provided an alternative proof, showing that NMAC is a PRF assuming *only* that h is a PRF. The assumption that H was CR was thus shown to be unnecessary. This helps explain why CR-finding attacks did not translate into attacks on NMAC or HMAC.

To get a sense of the numbers, we take a closer look at Bellare's result [11]. Continue to let $h\colon \{0,1\}^c \times \{0,1\}^b \to \{0,1\}^c$ be the underlying compression function. Let $\mathcal{A}$ be an adversary attacking the PRF security of NMAC. Assume $\mathcal{A}$ makes at most q oracle queries (assume $q \geq 2$), the i-th of which has length at most m_i ($1 \leq i \leq q$). Let $\sigma = m_1 + \cdots + m_q$ be the sum of the lengths of the queries and $m = \max(m_1, \ldots, m_q)$ be the length of a longest query. Then Bellare's result shows that there are adversaries $\mathcal{A}_1, \mathcal{A}_2$ such that

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathrm{HMAC}}(\mathcal{A}) \;\leq\; \mathbf{Adv}^{\mathrm{prf}}_{h}(\mathcal{A}_1) + (q-1)(\sigma - q/2) \cdot \mathbf{Adv}^{\mathrm{prf}}_{h}(\mathcal{A}_2) + \frac{q(q-1)}{2^{c+1}}\,. \qquad (9.1)$$

The running times of $\mathcal{A}_1, \mathcal{A}_2$ are about the same as that of $\mathcal{A}$. However, while $\mathcal{A}_1$ makes at most q oracle queries and $\mathcal{A}_2$ makes only two.

If t is a running time then let $\bar{t} = t/T_h$ where T_h is the time to compute h. Assume that the best attack against h as a PRF is exhaustive key search. (Birthday attacks do not apply since h is a family of functions, not a family of permutations.) This means that $\mathbf{Adv}^{\mathrm{prf}}_{h}(\mathcal{A}) \leq \bar{t} \cdot 2^{-c}$ for any prf-adversary $\mathcal{A}$ of time complexity t making $q \leq \bar{t}$ queries. Plugging this into (9.1) and simplifying, the upper bound on the prf-advantage of any adversary against NMAC that has time-complexity t and makes at most q queries is $O(\bar{t} + m^2 q^2 T_h) \cdot 2^{-c}$. If we ignore the T_h term, then this hits 1 when $q \approx 2^{c/2}/m$. This means that the bound justifies NMAC up to roughly $2^{c/2}/m$ queries.

9.5. Additional remarks on proven security. Define $\bar{h}\colon \{0,1\}^b \times \{0,1\}^c \to \{0,1\}^c$ by $\bar{h}(x,y) = h(y,x)$ for all $x \in \{0,1\}^b$ and $y \in \{0,1\}^c$. To say that $\bar{h}$ is a PRF is to say that h is a PRF when keyed by the message block rather than the chaining variable.

Bellare [11] points out that KDF0 can be proved a PRG under the assumption that $\bar{h}$ is a PRF under a form of related-key attack [20]. This assumption, however, is stronger than merely and directly assuming PRG security of the KDF, and this assumption has been shown to fail for MD5 [53]. Accordingly, we recommend the PRG assumption on the KDFs, which still stands, even for MD5. The difference is that the attack relies on having values of $h(x, K)$ for chosen values of x, where K is the key, while the PRG assumption fixes x to be IV.

There are a few other security proofs that aim to establish MAC (but not PRF) security of NMAC under assumptions that are different from, or weaker than, the PRF assumption on the compression function [11, 69]. These provide fallback guarantees for MAC security in the case the compression function is found to *not* be a PRF.

Mode	Type	q	t	RKA?	Who
NMAC-MD4	recover $K1, K2$	2^{77}	2^{77}	No	[196]
HMAC-MD4	DH	2^{58}		No	[53]
HMAC-MD4	recover $K1$	2^{63}		No	[53]
HMAC-MD4	forgery	2^{58}		No	[111]
HMAC-MD4	recover $K1, K2$	2^{72}	2^{77}	No	[196]
NMAC-MD5	recover $L1$	2^{47}		Yes	[53]
NMAC-MD5	recover $L1 \parallel L2$	2^{51}	2^{100}	Yes	[70]
NMAC-MD5	recover $L1 \parallel L2$	2^{76}	2^{77}	Yes	[70]
HMAC-MD5	DH	2^{97}	2^{97}	No	[200]

Figure 9.3: **Attacks on NMAC and HMAC.** The HMAC versions to which they apply would appear to be 0 and 1. We indicate the type of attack, the number of queries q, the running time t in hash computations, whether or not it is a related-key attack (RKA), and the source. The advantage is in all cases at least 0.5. Blanks indicate we do not know.

As the above indicates, security proofs view the compression function $h\colon \{0,1\}^c \times \{0,1\}^b \to \{0,1\}^c$ as the base primitive. Typically (in particular, for MD5 and SHA1), the function h is built from an underlying blockcipher $E\colon \{0,1\}^b \times \{0,1\}^c \to \{0,1\}^c$ in Davies-Meyer modes [138], so that $h(x,y) = E_y(x) \oplus x$. It is thus natural to consider basing the security of NMAC on some assumption on the blockcipher. The most attractive would certainly be the standard assumption that E is a PRP.

That said, the PRF property of h does *not* follow from the PRP assumption on E. We do not merely mean that nobody knows how to do this but that it is impossible because one can give an example of a PRP E for which h is not a PRF. Intuitively, the difficulty is that the key for E is y, which is under the attacker's control when attacking h as a PRF, while the key x for h is used as a data input for E. One might ask, then, whether there is an alternative, direct security proof for NMAC based on the PRP assumption on E but this again would not appear to be possible. What *is* true, however, is that if E is modeled as an ideal cipher then h will be a PRF [41]. Thus NMAC is secure in the model where E is an ideal cipher.

9.6. Attacks. When the hash function is iterated, HMAC is an instance of what Preneel and Van Oorschot call an iterated MAC [172]. Their birthday attack then applies to break HMAC as a MAC with about $2^{c/2}/\sqrt{m}$ queries of at most m blocks each. This means the security guaranteed by the above proof is close to best possible (the two differ by a factor of $\sqrt{m}$).

The birthday attack is generic, meaning that it applies equally well for any hash function, be it MD5, SHA1, or something else. Cryptanalysts have also attempted to find dedicated attacks, meaning to break HMAC for particular choices of H. The terrain is complex, with attacks of many different kinds, but the bottom line is that none of them currently threaten HMAC-MD5 or HMAC-SHA1 relative to the central PRF goal, and none appear to refute the assumption that the compression functions underlying MD5 or SHA1 are PRFs.

Taking a closer look, we attempt to summarize known cryptanalytic attacks in Figure 9.3. Let us now discuss the Figure and its implications.

The entries in the attack-type column should be self-explanatory except for the label DH. This stands for "Distinguishing-H." Here one attempts to distinguish the construct (NMAC or

HMAC), with the stated hash function, from the same construct with a random compression function. The interest of this model is apparently that birthday attacks do not apply and hence beyond-birthday cryptanalytic attacks are cryptanalytically interesting. From an application perspective, however, the goal is moot, and if one views the security goal as being a PRF, as we think one should, none of the shown DH attacks are significant, since they do not improve on the 2^{64} time birthday attack.

Even for MD4, the attacks are beyond birthday. It is potentially significant that beyond-birthday effort can recover the key, not just forge. Some attacks only recover half the key, which would not appear damaging.

Many attacks on NMAC are related-key attacks (RKAs), meaning they attack NMAC when it is being used simultaneously with different keys related in attacker-chosen ways. Related-key attacks are standard in the cryptanalytic literature, but related keys are rare and frowned upon in "correct" usage of a PRF, making the practical utility of such attacks moot. Also, it is unclear how to lift RKAs on NMAC to HMAC. Paradoxically, the reason is that in HMAC, the keys $K1, K2$ are related and derived from K in some particular way. We clarify that this internal use of related keys is very different from external use, where the issue is several instances of HMAC, each with its own key K, but these keys are related. This simply should not happen when HMAC is appropriately used.

We have not discussed attacks involving reduced-round hash functions, of which many are known [111, 120, 173].

9.7. Truncation. HMAC truncated to τ bits is, inevitably, subject to an attack that forges with probability $q_v/2^\tau$ in q_v verification attempts; as a consequence, the parameter τ should, in most cases, be kept fairly large. Why, then, truncate at all? One reason for allowing MAC truncation is the obvious one, to lessen bandwidth or storage costs. More interestingly, truncation appears to make the birthday attack less effective, because internal collisions can no longer be identified by collisions in the HMAC outputs. A little truncation may thus improve security. We emphasize, however, that while there are proofs that truncation will not degrade security by more than the addition of the term that we just mentioned, there is *no* known proof that truncation can *improve* security—not even in the random-oracle model. We suspect, however, that this is more indicative of technical difficulties in the proof—or the lack of a really serious effort—than the absence of a quantifiable improvement.

We emphasize that the concrete security loss from truncation is "only" the $q_v/2^\tau$ term that one "must" be for truncation; we are not speaking of the more severe drop in security as one sees when truncating tags with GMAC (§10.6) or GCM (§12.6).

9.8. Beyond-PRF goals. HMAC has been shown to have good randomness-extraction properties in the model where the compression function is random [59, 72]. This supports its use for key-derivation. The security of HMAC under various kinds of side-channel attacks has been considered in [130, 161].

9.9. Perspective. We have discussed both proofs and attacks, outcomes from very different communities with very different cultures. Lacking in the literature is anything that helps understand how they relate. We aim here to provide a bit of perspective on this.

Recall that HMAC is a secure PRF assuming the underlying compression function h is a PRF [11]. From the proof perspective, the natural next step is to validate or refute this assumption for specific hash functions. We would look to cryptanalysts to cryptanalyze the

compression functions of MD4, MD5, and SHA1 as PRFs.

Cryptanalysis has, indeed, shown that md4, the compression function of MD4, is not a PRF, indicating we should drop HMAC-MD4 even in the absence of direct attacks on it. Beyond this, however, cryptanalysts seem to attack HMAC itself, directly, for the above or other choices of hash functions, rather than attack the compression function. When they fail to break it as a PRF, we have direct evidence of the PRF security of HMAC, but also indirect evidence that the compression functions are PRFs.

Cryptanalysis of HMAC shows a trend common to other primitives. When breaking relative to the main target goal (here, PRF) is hard, the models are extended until attacks become possible. Thus, we see the distinguishing-H attacks noted above. Alternatively, attackers are allowed more capabilities. Thus we see related-key attacks. The cryptanalytic perspective is perhaps that finding attacks in variants of the model of interest may help to find "real" attacks later. If so, for HMAC, it has not yet borne fruit.

Why do cryptanalysts attack HMAC rather than the compression functions? Perhaps it is just a difference in cultures. Cryptanalysts gain confidence in a design only via a history of failed attacks, and so attacking the design is their goal. But one might also note that while the compression function being a PRF is sufficient for PRF security of HMAC, it is not necessary, in the sense that HMAC may be a PRF even if its compression function is not. Put another way, suppose tomorrow someone finds an attack on the compression function of MD5, breaking it as a PRF. This may break HMAC-MD5 — or it may not. This may contribute to cryptanalytic interest focusing on HMAC rather than the compression function. Cryptanalysts are after the bigger prize. By way of analogy, collisions were found in the compression function of MD5 some time ago [57]. This nullified the proof-based guarantees for the collision-resistance of MD5, yet had little impact since it did not yield MD5 collisions. Attacks on MD5 itself got more attention and prestige.

9.10. Conclusions. HMAC remains strong, as we find that compression functions that are not collision-resistant still appear to be good PRFs, which is enough to guarantee HMAC's security. Still, caution should be exercised. HMAC-MD4 should be avoided (no effective attacks are known, but the compression function is not a good PRF). Schemes HMAC-MD5 and HMAC-SHA1 are holding up well, but HMAC-SHA256 is seen as a more conservative choice for long-term security. Beyond that, we look to HMAC-SHA3.

Basing a MAC on an unkeyed cryptographic hash function—what HMAC looks to be doing—might seem like a conceptually wrong turn. First, the fundamental property of a cryptographic hash function—collision resistance—is not a good basis for making a MAC [189]. Second, mechanisms like the "envelope" approach for turning a hash function into a MAC [193]—the kind of construction from which HMAC evolved—can be seen, historically, as "engineering" attempts to avoid US export controls and maximize the utility of a single, well-known tool. But HMAC need not be viewed as being based on an unkeyed cryptographic hash function: instead, it is based on a keyed compression function. A keyed compression function is a good starting point for creating a MAC/PRF. In some sense, the troublesome step in hash-function design has always been in getting *rid* of the secret key. It is here that problems arise, as suggested by the fact that MD5 has known collisions, but its keyed compression function is perfectly fine. Regarding HMAC as transforming a keyed compression function into a variable-input-length PRF, the mode avoids the elimination (and re-insertion) of a cryptographic key.

Basing a MAC on a cryptographic hash function seems to place significant limits on the speed it can be expected to deliver. Traditionally, MD4-family cryptographic hash functions were seen as being faster, in software, than blockciphers. But this is no longer an accurate

view, for two reasons. First, attacks on cryptographic hash functions have steadily pushed us towards selecting more conservative, less efficient designs. As these get incorporated into HMAC—the evolution of HMAC-MD4 $\mapsto$ HMAC-MD5 $\mapsto$ HMAC-SHA1 $\mapsto$ HMAC-SHA2 $\mapsto$ HMAC-SHA3—speed gets worse.[1] The turmoil in hash functions has nothing to do with HMAC, but it ends up going along for the ride, and in some sense suffering for it. Beyond this, as AES gets increasingly provided with hardware support, the time to compute an AES-based MAC improves, but HMAC is not positively impacted. As a current snapshot, the fastest reported time for SHA1 on an Intel x86 processor is 5.8 cpb [125]. In contrast, CBC encryption, and therefore the CBC MAC and its relatives, runs at about 4.1 cpb on a recent Intel x86 processor [162]. And the 30% difference in speed understates the discrepancy: the coarse granularity of HMAC (the fact that, minimally, one must perform two compression functions, each on 64 bytes) means that the x86 cost for HMAC-SHA1 will start at about 700 cycles, whereas the AES CBC-MAC of a short string, in the presence of hardware AES support, will take just a handful of cycles. In brief, the original conception of HMAC—"build a MAC out of a cryptographic hash function in a black-box manner"—is a conception that has lead to a popular and well-designed construction, but it is not the most efficient approach.

[1] Of course the speed impact of the final step in this chain remains unknown.

Chapter 10

GMAC Mode

10.1. Summary. The Galois/counter MAC (GMAC) is a nonce-based message authentication code that is defined alongside of the authenticated-encryption scheme GCM (Chapter 12); it is a special case of that mode [64]. Specifically, GMAC is defined by asserting, in the notation that we will adopt, that

$$\mathrm{GMAC}_K^N(M) \quad = \quad \mathrm{GCM}_K^{N,M}(\varepsilon). \tag{10.1}$$

In words, the MAC of a string $M \in \{0,1\}^*$ relative to the key K and nonce N is the GCM-encryption of the empty string under the same key and nonce, but M as the associated data.

As a special case of GCM, GMAC inherits many of the latter mode's strengths and weaknesses, and much of what is said about GCM in Chapter 12 applies *ipso facto* to GMAC. Still, there are a few specific things that should be said with respect to (just) GMAC, and this chapter says those things.

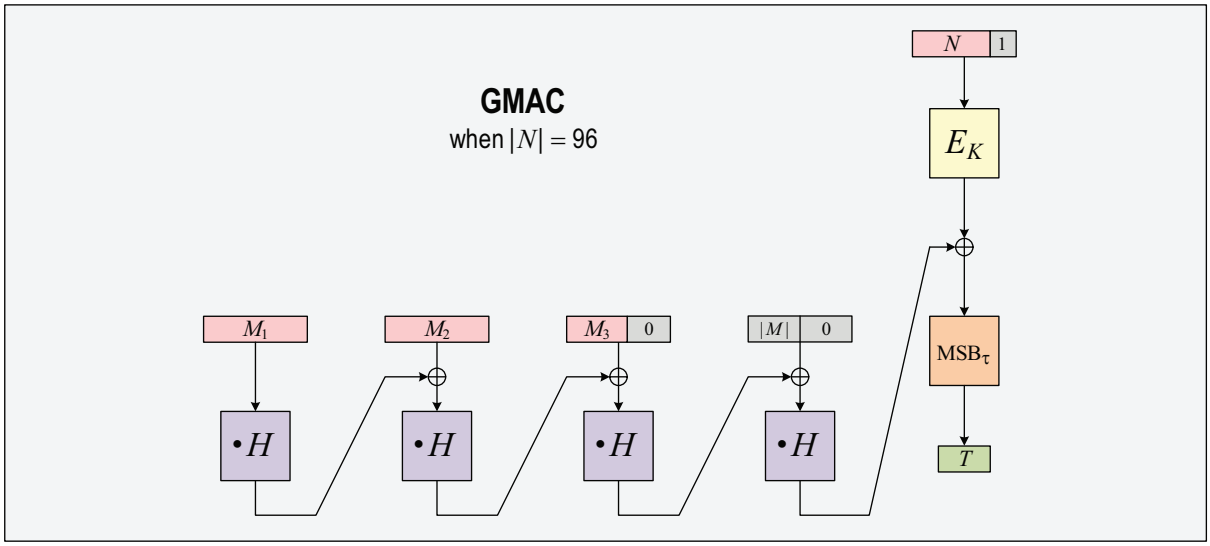
GMAC is a very different MAC from all the others reviewed in Part II of this report: it is the only MAC that is *nonce-based*. Cryptographers don't usually speak of nonce-based MACs, but we sometimes speak of stateful ones, which are almost the same thing. The difference is that, in a nonce-based MAC, the user, not the mechanism, is responsible for maintaining the state, and all that the users is expected to do is to ensure that the state—the nonce—is ever-changing.

Why did NIST decide to give GMAC its own name and individual stature, instead of just implicitly defining it as a special case of GCM? The answer is hinted at in [64, Section 10]: GMAC is considered as a separate "category" of an object that can claim conformance to the NIST standard. As such, a GMAC implementation may, apparently, be separately validated under FIPS Publication 140-2 [157].

GMAC has several things going in its favor—mostly the same things as with GCM. Still, I am rather more critical of explicitly standardizing it.

10.1.1 GMAC is stateful (or nonce-based), where MACs need not be, creating opportunity for misuse. This would be OK if there were significant benefits from the included nonce. But, in this case, there are not.

10.1.2 While there *should* be a provable-security benefit—better bounds—by virtue of the nonce, GMAC largely squanders this benefit, delivering bounds that are sometimes even worse than what one expects to get from a conventional deterministic MAC. One could prove strong bounds if $|N| = 96$ and there is no truncation of the MAC, but such bounds have not be explicitly proved [133].




```
10  algorithm GMAC^N_K(M)                                              GMAC
11  // |M| < 2^64,  |N| > 0, |N| < 2^64,  M ∈ ({0,1}^8)*,  N ∈ ({0,1}^8)*
12  H ← E_K(0^128)
13  if |N| = 96 then Cnt ← N ‖ 0^31 1
14               else Cnt ← GHASH_K(N ‖ 0^i ‖ |N|_128) for minimal i ≥ 0 s.t. 128 | (|N| + i)
15  Y_1 ← E_K(Cnt)
16  X ← M ‖ 0^i ‖ |M|_64 ‖ 0^64 for minimal i ≥ 0 s.t. 128 | (|M| + i)
17  Tag ← Y_1 ⊕ GHASH_H(X)
18  T ← MSB_τ(Tag)
19  return T

20  algorithm GHASH_H(X)                                          Used internally
21  X_1 ··· X_m ← X where |X_i| = 128
22  Y ← 0^128;   for i ← 1 to m do Y ← (Y ⊕ X_i) • H
23  return Y
```

Figure 10.1: **Definition of GMAC.** The mode depends on blockcipher $E: \{0,1\}^k \times \{0,1\}^{128} \to \{0,1\}^{128}$ and the tag length $\tau \in \{32, 64, 96, 104, 112, 120, 128\}$. The illustration is specific to the case of $|N| = 96$; longer or shorter nonces are processed as specified in the code.

10.1.3 Tag-truncation in GMAC has the same problems as tag-truncation in GCM; forgeries can be accomplished after about $2^{\tau/2}$ queries, or 2^τ total blocks. Correspondingly, it is essential to obey NIST's (rather severe) restrictions when using GMAC to generate 32-bit or 64-bit nonces.

10.1.4 There is a well-entrenched expectation and tradition for truncating MACs down to 32-bits, which has, after all, been the standard in retail banking for decades. A general-purpose MAC should provide the expected service when so truncated.

10.1.5 The idea of permitting arbitrary-length nonces seems to make still less sense with GMAC than it did with GCM.

10.1.6 One cannot say that these accommodations were necessary for speed; the software speed of GMAC is nothing special compared to alternative constructions.

All in all, the gap between what GMAC delivers and what a "good" MAC should deliver feels larger than the gap between what GCM delivers and what a "good" AEAD scheme should

deliver, which is why I am more critical of it.

I believe that GMAC is a good choice for an application if, for that applications, you have already implemented GCM, will not be be truncating the MAC, don't need to minimize the MACs length, are quite certain of the nonce property of the provided IVs, and want to minimize the footprint of any additional code. This is a rather specific set of circumstances.

10.2. Definition of the mode. GMAC is defined and illustrated in Figure 10.1. The mode is parameterized by a 128-bit blockcipher $E\colon \{0,1\}^k \times \{0,1\}^{128} \to \{0,1\}^{128}$ and a tag length $\tau \in \{32, 64, 96, 104, 112, 120, 128\}$. Following SP 800-38D [64], the underlying blockcipher *must* be AES. The requirement follows from assertions that the blockcipher must be NIST-approved and have a 128-bit block size.

Quantity-of-use restrictions are placed on GMAC if using either of the smallest two permitted tag lengths [64, Appendix C]. Specifically, for 32-bit tags, an implementation must ensure that when the maximal permitted message length is 2^5 (or 2^6, 2^7, 2^8, 2^9, or 2^{10}) bytes, the maximal permitted number of applications of the MAC-verification function does not exceed 2^{22} (or 2^{20}, 2^{18}, 2^{15}, 2^{13}, 2^{11}, respectively). Limits are not specified if the maximal permitted message length exceeds 2^{10} bytes; we assume that such messages must not be allowed. Similarly, for 64-bit tags, if the maximal permitted message length is 2^{15} (or 2^{17}, 2^{19}, 2^{21}, 2^{23}, or 2^{25}) bytes, the maximal permitted number of applications of the MAC-verification function must be no more than 2^{32} (or 2^{29}, 2^{26}, 2^{23}, 2^{20}, 2^{17}, respectively). Limits are not specified if the maximal permitted message length exceeds 2^{25} bytes; we assume that such messages are disallowed.

10.3. Desirable characteristics. Many of the nice features of GCM are shared by GMAC: provable security guarantees; only the forward direction of the blockcipher being used; unsurpassed hardware performance; parallelizability (although it is not so obvious from the chained description); reasonable software efficiency; on-line status (one does not need to know the length of M before authenticating it); ability to precompute the (one) AES call if the next nonce is known; incrementality with respect to appending and substituting 128-bit blocks; hardware support in "Westmere" (and later) Intel processors via PCLMULDQ; and general absence of patents by the scheme's inventors. I will not dwell on any of these benefits; see §12.3.

10.4. Nonce-based message authentication. In §II.2 we defined MACs as stateless, deterministic functions: given a key K and a message M, we produce a tag $T = F_K(M)$ that depends only on these two things. There is also a tradition for *stateful* MACs. One usually conceives of them as *pair* of function, a MAC-generation algorithm and a MAC-verification one, the first of which *generates* a tag T and updates its internal state, and the second of which, which is stateless, *verifies* a tag T for a message M. The tag would include any necessary state information. Both algorithms take in the underlying key.

We will describe a variant of the above where the state is provided by the user and is only expected to be unique across the different calls; this is *nonce-based* message authentication. Here it is adequate to consider a single algorithm, Mac, as with a conventional MAC. Algorithm Mac takes in three arguments—a key K, a nonce N, and a message M—and returns an τ-bit string. Function Mac will have signature $\mathrm{Mac}\colon \mathcal{K} \times \mathcal{N} \times \mathcal{M} \to \{0,1\}^\tau$ and we'll write the arguments as in $\mathrm{Mac}_K^N(M)$.

To define security for a nonce-based MAC, we'll carry out the following experiment. We begin by choosing a random key $K \in \mathcal{K}$ from the key space. We now give the adversary two oracles, a MAC-generation oracle and a MAC-verification oracle.

- The **MAC-generation oracle**, on a query of $(N, M) \in \mathcal{N} \times \mathcal{M}$, returns $T = \mathrm{Mac}_K^N(M)$. The adversary is not allowed to repeat a nonce-value (an N) in any of these queries.
- The **MAC-verification oracle**, on a query of $(N, M, T) \in \mathcal{N} \times \mathcal{M} \times \{0,1\}^\tau$, returns the bit 1 if $\mathrm{Mac}_K^N(M) = T$, and the bit 0 otherwise. There are no restrictions about what nonce-values (N-values) may be used.

While the phrase "nonce-based MAC" is not routinely used, the notion is not really new. The standard Carter-Wegman MAC [50, 201], as it has been adapted to the complexity-theoretic setting [46, 177], can be considered as a nonce-based MAC. Here the idea is to start with an almost-xor-universal hash-function family (ϵ-AXU) [116] $\mathrm{Hash}\colon \mathcal{K} \times \mathcal{M} \to \{0,1\}^n$ and a pseudorandom function $\mathrm{Prf}\colon \mathcal{K}' \times \mathcal{N} \to \{0,1\}^n$. Saying that Hash is ϵ-AXU means that, for all distinct $M, M' \in \mathcal{M}$ and all $\Delta \in \{0,1\}^n$, we have that $\Pr_K[H(M) \oplus H(M') = \Delta] \leq \epsilon$. One defines the MAC from Hash and Prf by setting

$$\mathrm{Mac}_{K,K'}^N(M) \quad = \quad \mathrm{Prf}_{K'}(N) \oplus H_K(M)\ . \tag{10.2}$$

Typically, we let Prf be a blockcipher E.

This Carter-Wegman construction, as above, is close to what is done in GMAC, but there are a couple of important differences. The first is that, in GMAC, the keys K and K' are not independent. The second is that, when the nonce N is not 96 bits, the pseudorandom function Prf is not simply the blockcipher E. Instead, it is a rather complex construction that involves applying a polynomial-based hash to a padded and length-annotated N, keyed by the same key that is used for the hash Hash, and *then* applying the blockcipher E.

10.5. Provable security. The paper on GCM offers no results specific to GMAC; instead, the scheme inherits its provable-security claim from GCM [133]. We explain what that claim would be. Let $\mathcal{B}$ be an adversary that asks its oracle a sequence of queries where

- ℓ upperbounds the block length for each and every message that will be authenticated, and
- ℓ_N upperbounds the block length of each and every nonce that will be used,
- q upperbounds the total number of queries (either MAC-generation or MAC-verification queries).

Then McGrew and Viega's [133, Theorem 2] implies that

$$\mathbf{Adv}^{\mathrm{auth}}_{\mathrm{GCM}[\mathrm{Perm}(n),\tau]}(\mathcal{B}) \quad \leq \quad \frac{q^2\ell_N + 3q^2 + 0.5q\ell_N + 0.5q}{2^n} + \frac{q(\ell+1)}{2^\tau}\ . \tag{10.3}$$

Simplifying and giving up a constant, for readability, we have that

$$\mathbf{Adv}^{\mathrm{auth}}_{\mathrm{GCM}[\mathrm{Perm}(n),\tau]}(\mathcal{B}) \quad \leq \quad \frac{5q^2\ell_N}{2^n} + \frac{q(\ell+1)}{2^\tau}\ . \tag{10.4}$$

This is not a good bound, for two reasons:

10.5.1 The first problem is that the addend in the first term of (10.4) is growing in $\sigma^3/2^n$, where σ is the total number of blocks that adversary asks during the attack. (To get this worst-case behavior, given a budget of σ blocks, use half of them for a single long nonce with the empty message, and use the rest with one-block messages and a one-block nonces). In any modern MAC, we want and expect to see security results no worse than $\sigma^2/2^n$.

10.5.2 The second problem is the quadratic behavior of the second addend. The purpose of a stateful MAC, since at least Bellare, Guérin, and Rogaway [17], is to get security that degrades only linearly in the number of queries. More specifically, when using the Carter-Wegman construction with an ϵ-xor universal hash function, if q_{ver} verification queries are made by the adversary, then its chance to forge is going to be bounded by $\epsilon\, q_{\text{ver}}$, something linear in q_{ver}, if we think of ϵ as a small constant.

Now one might answer: "*Right.* And here $\epsilon = (m+1)2^{-\tau}$, the value one gets from polynomial hashing." But this is *not* the value one would want or expect to see from a truncated-to-τ-bit hash function: one wants to see a bound for ϵ that is something like $\epsilon = 2^{-\tau} + (\ell+1)2^{-n}$. It may sound like a technically-insignificant detail, but it is not: in the one case, truncation by 32-bits is reasonable and carries with it a fine guarantee; in the other case, this is not so.

While issue 10.5.1 (the $\sigma^3/2^n$ issue) is probably just a bit of sloppiness in the analysis, issue 10.5.2 (short tags) is not; it's a problem with the scheme. If the authors were envisioning allowing truncation they should have processed the polynomial-hash differently, for example, composing it with a variationally-universal hash function [117], as Krovetz and Rogaway taught, or composing it with a variationally-xor-universal hash function, as Wang, Feng, Lin, and Wu have more recently explained [197].

In conclusion, there *are* provable-security bounds for GMAC, but they are weaker than what one wants and can reasonably expect.

10.6. Forging GMAC with truncated tags. An attack by Ferguson [67] on GCM with 32-bit tags asks for the encryption of a single 2^{17}-block message (and associated data of the emptystring) and then, with an expected 2^{16} forgeries, finds a valid 32-bit tag. Soon after, the subkey H is completely compromised, and the adversary can forge anything.

While Ferguson does not point this out, his attack can be adapted to attack GMAC. In that setting, the method would work like this.

Begin by selecting an arbitrary nonce N and message $M = a_m \cdots a_3 a_2$ consisting of $m = 2^{17} - 1$ blocks $a_i \in \{0,1\}^{128}$. After padding, this will become a 2^{17}-block string $a_m \cdots a_2 a_1$ where a_1 encodes $|M|$. For concreteness, we take all of the blocks indexed a_j, where $j \geq 3$, to be zero; the other 17 blocks can be 1, say, or can be selected at random. Note that when we hash the message M in GMAC by evaluating the polynomial $g_M(x) = a_m x^m + \cdots + a_1$ at H, the observation that $x \mapsto x^2$ in $\text{GF}(2^{128})$ is *linear* when regarded as an operation on bit vectors implies that $g_M(H) = A_M \cdot H$ can be regarded as the multiplication of the 128-bit vector H by a 128×128 binary matrix A_M that the adversary can compute.

Let the adversary ask the oracle for $T = \text{GMAC}_K^N(M)$. We now seek some alternative message B having the same length and structure as M such that $\text{GMAC}_K^N(B)$ is also T. For each such alternative string B we will do a verification query with message B and unchanged N and T. We will succeed if the padded-B hashes to something that has the same first 32-bits as what the padded-M hashed to. Saying that B has the same structure as M means that we will select B to likewise have the form $B = b_m \cdots b_2$ where b_i is zero for nonzero powers of i. We have 17×128 random bits to play with in choosing B.

Now for any candidate B we can once again compute the matrix A_B such that message B hashes to $A_B \cdot H$ prior to xoring it with with $E_K(N \parallel 0^{31}1)$. We hope that $A_B \cdot H$ and $A_M \cdot H$ agree on their first 32 bits, which is to say we would like to find B-values such that $(A_B - A_M) \cdot H$ always begins with 32 zeros, regardless of H. This will be ensured if we select B such that the matrix $\Delta_B = A_B - A_M$ begins with 16 zero rows. Well, we have 128×16 constraints we would like to satisfy and we have 128×17 variables at our disposal, so we can sample in the space of random B-vectors such that Δ_B begins with 16 zero-rows.

When we do so, we expect to forge in an expected 2^{16} queries.

A bit more generally, the method above lets us forge τ-bit GMAC tags in an expected $2^{\tau/2}$ queries, each of $2^{\tau/2+1}$ blocks. The total number of blocks used is thus about $\sigma = 2^{\tau+1}$. This is not excessive when τ is small.

The attack demonstrates that issue 10.6.2 is a "real" problem, not just a gap between what is proven and what is true.

As I have indicated, there are reasonably cheap ways to "fix" the short-tag problem; for example, one could have multiplied the polynomial-hash result by another, independent field point [197]. But such suggestions come too late for GMAC or GCM, which are stable, standardized schemes.

I believe that, for MACs, one *ought* to be able to truncate down to 32-bits, and even below. It is not just that retail banking has used 32-bit tags for decades, finding tags of that length, properly created, to be adequate for protecting billions in transactions. The fact is that human beings routinely take "risks" that are orders of magnitude larger than 2^{-32}. Such risks are a necessary and sensible thing to do. It is worth keeping in perspective that ones chance of being struck by lightning, in a given year, is perhaps 1000 times greater than 2^{-32}. Escalation of computing speed is typically *irrelevant* in ascertaining how long MACs should be, while network bandwidth is only sometimes a relevant factor.

10.7. Against nonce-based MACs. Returning to one of the criticisms voiced in §10.1, I comment that there is relatively little precedent for stateful MACs. The only other standardized MAC that I know is the NESSIE-standard UMAC scheme [37]. But in that case the use of the nonce was strongly tied to improvements in speed and bounds compared to what a comparable stateless scheme could deliver. For GMAC, there are no such gains. And while encryption schemes *need* nonces, state, or randomness to achieve strong security goals, MACs simply do not, and so there *should* be a clear benefit for demanding a nonce. I would note too that the frequency of misuse of the IV in CBC encryption (people using a fixed, counter, timestamp, or last-prior-block-of ciphertext IV) is already suggestive that IVs are subject to misuse. And here, IV reuse is catastrophic: for the reasons already described by Joux in the context of GCM, but which are no less applicable here, GMAC will completely fail when an IV is used twice [104]. Authenticity will be forfeit from that point on.

In the end, we "forgave" GCM its various deficiencies and suggested that it is fine to standardize it. I am less charitable with GMAC, because of its "unnecessary" requirement for a nonce, its failure to use it well, and the problems it encounters creating safe, short tags.

Part III

Authenticated-Encryption Modes

Part III

Authenticated-Encryption Modes

III.1. Background. Over the past few years, considerable effort has been spent to construct, employ, and standardize cryptographic schemes for *authenticated encryption* (AE). One reason for this is recognition of the fact that a scheme that delivers both privacy *and* authenticity may be more efficient than the straightforward amalgamation of separate privacy and authenticity techniques. A second reason is the realization that an AE scheme is less likely to be incorrectly used than an encryption scheme designed to deliver privacy alone.

While other possibilities exist, it is natural to build AE schemes from blockciphers, employing them in some mode of operation. There are two approaches. In a *composed* AE scheme one conjoins essentially separate privacy and authenticity modes. For example, one might apply CTR-mode encryption and then compute some version of the CBC MAC. In an *integrated* AE scheme the parts of the mechanism responsible for privacy and for authenticity are tightly coupled.[1] Composed AE schemes are rooted in cryptographic folklore, with the first systematic study carried out by Bellare and Namprempre [22]. Integrated schemes emerged about a decade ago, with the work of Jutla [106, 107], Katz and Yung [109], and Gligor and Donescu [73].

The two AE schemes considered within this evaluation report, CCM [62, 203] and GCM [64, 137], are both composed schemes. The first is built from CTR mode and the CBC-MAC, the second, from CTR mode and a Carter-Wegman MAC—one created from the universal hash function that is polynomial evaluation over $\mathrm{GF}(2^{128})$ [50, 201].

Both CCM and GCM are *nonce-based*, meaning that, on encryption or decryption, one must present a nonce, N. When encrypting, the nonce must be changed with each message. A counter of some sort would most typically be used, although it is not necessary that one understands "increment" as incrementing in the realm of integers. On decryption one must present the same nonce N that was used to encrypt the message. We do not consider the nonce to be part of the ciphertext. How it is communicated to the party that will decrypt is not the business of the formalization; the nonce might be flowed with the ciphertext, or it might be communicated by some other means, like the sender and receiver maintaining some state, like a counter, that is kept in-step.

Both CCM and GCM encryption take as input not only the key, nonce, and plaintext but, also, a string called the *associated data* (AD). The AD is authenticity-protected by the ciphertext. It is not included as part of the ciphertext and is not expected to be recoverable from it. Sometimes one sees the acronym "AEAD"—authenticated-encryption with associated-data—to emphasize the presence of the AD. In this report, however, AE *means* AEAD; the AD

[1] Integrated and composed schemes are sometimes called "one-pass" and "two-pass" schemes. These names are misleading, however, as both types of schemes can typically be implemented in a single pass over that data.

is always provided for. A typical use of the AD is to encode header information in a networking context. Non-changing parts of the header should be authenticated but must not be encrypted, since the information will be needed for routing at intermediate nodes that do not have the underlying key. The first formalizations for AE in the presence of AD is due to Rogaway [178], but the notion of AD, and some ways of dealing with it, our clearly folklore.

Ciphertexts produced by an AE scheme must be sparse in the sense that "most" ciphertexts should decrypt to INVALID. When the receiver receives a ciphertext C and, using the key K, nonce N and associated data A, it decrypts to some string M—a value other than INVALID, the receiver can take this as proof—or at least strong evidence—that this ciphertext C actually *was* produced and sent by the sender, the result of encrypting its plaintext M using the nonce N and AD A.

In CCM, GCM, and other AE schemes, the ciphertext C has the structure $C = C \parallel T$ for a fixed-length tag T, $|T| = \tau$. Given the key K, nonce N, and AD A the receiver can recover from C, the *ciphertext core*, a *prospective* plaintext M and an *anticipated* tag T'. The receiver accepts C as authentic, having corresponding plaintext M, if the received tag is the anticipated one, $T = T'$. Otherwise, the receiver is expected to return INVALID, ignoring the prospective message M. We call an AE scheme of this structure a *tag-based* one. CCM and GCM are tag-based AE schemes.

We comment that, while CCM uses the term "ciphertext" in the way we have, encompassing both the ciphertext core and the tag, the GCM specification uses the term "ciphertext" to mean just the ciphertext core.

III.2. Definition of authenticated encryption. We provide a formalization for (nonce-based, AD-employing) AE. We conceptually follow [178, 182], but we substantially revise the "syntax" of an AE scheme, in order to increase the family-resemblance to our treatment of blockciphers and IV-based encryption schemes.[2]

A scheme for (nonce-based) authenticated encryption (with associated-data) is a function

$$\mathcal{E}\colon \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{X} \;\to\; \mathcal{X}\,. \tag{10.5}$$

Nonempty sets $\mathcal{K}$, $\mathcal{N}$, $\mathcal{A}$, and $\mathcal{X}$ are called the *key space*, *nonce space*, *AD space*, and *message space*. We assume $\mathcal{N}, \mathcal{A}, \mathcal{X} \subseteq \{0,1\}^*$ are all sets of strings. The key space is either finite or otherwise endowed with a probability distribution. The encryption function takes in a key $K \in \mathcal{K}$, a nonce $N \in \mathcal{N}$, a string of associated data $A \in \mathcal{A}$, and a plaintext $P \in \mathcal{X}$. It returns a ciphertext $C = \mathcal{E}_K^{N,A}(P) = \mathcal{E}(K, N, A, P) \in \mathcal{X} \subseteq \{0,1\}^*$. We demand that, for all K, N, A, the function $\mathcal{E}_K^{N,A}(\cdot)$ is injective. Given this requirement, we may associate to any AE scheme $\mathcal{E}$ a unique decryption function $\mathcal{D} = \mathcal{E}^{-1}$ with signature

$$\mathcal{D}\colon \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{X} \;\to\; \mathcal{X} \cup \{\text{INVALID}\} \tag{10.6}$$

and defined by saying that $\mathcal{D}_K^{N,A}(C) = P$ if $C = \mathcal{E}_K^{N,A}(P)$ for some P, while $\mathcal{D}_K^{N,A}(C) = \text{INVALID}$ if there is no $P \in \mathcal{X}$ such that $C = \mathcal{E}_K^{N,A}(P)$.

We make the following assumptions on any AE scheme $\mathcal{E}$: that $|\mathcal{E}_K^{N,A}(P)| = |\mathcal{E}_K^{N,A}(P')|$ if $|P| = |P'|$. If this value is always $|P| + \tau$, as it is for CCM or GCM, we call τ the *tag length* of the scheme.

[2] In particular, AE schemes have normally been defined as a triple $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ consisting of a key-generation algorithm, an encryption algorithm, and a decryption algorithm. We here select a simpler syntax, where the scheme is specified only by the encryption function $\mathcal{E}$.

While we have defined an AE scheme as a function $\mathcal{E}$, any practical AE scheme will need to have an efficient algorithm for implementing it, as well as one for implementing its inverse $\mathcal{D}$. When we speak of computational complexity matters associated to an AE scheme, we must have in mind a particular algorithmic realization.

We now formalize security of an AE scheme $\mathcal{E}$, beginning with **privacy**. Given an adversary (an oracle-calling algorithm) $\mathcal{A}$, we let

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathcal{E}}(\mathcal{A}) \quad = \quad \Pr\left[K \xleftarrow{\$} \mathcal{K} : \ \mathcal{A}^{\mathcal{E}_K(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot,\cdot,\cdot)} \Rightarrow 1\right] \tag{10.7}$$

where queries of $\$(N, A, P)$ return a uniformly random string of length $|\mathcal{E}_K^{N,A}(P)|$. In other words, the adversary is given either a "real" encryption oracle or else a "fake" encryption oracle that simply returns the same number of bits that the real encryption oracle would return. We demand that $\mathcal{A}$ never ask two queries with the same first component (the N-value); such an adversary is said to be *nonce-respecting*. We also demand that the adversary never ask a query outside of $\mathcal{N} \times \mathcal{A} \times \mathcal{X}$, and that it never repeats a query. The notion we have defined is termed *indistinguishability from random bits*. It is just like the IND$-notion dealt with earlier in this report, the only real difference being the addition of the AD.

Next we define **authenticity** for an AE scheme $\mathcal{E}$ having inverse $\mathcal{D}$. For that, let

$$\mathbf{Adv}^{\mathrm{auth}}_{\mathcal{E}}(\mathcal{A}) \quad = \quad \Pr\left[K \xleftarrow{\$} \mathcal{K} : \ \mathcal{A}^{\mathcal{E}_K(\cdot,\cdot,\cdot),\mathcal{D}_K(\cdot,\cdot,\cdot)} \text{ forges}\right] \tag{10.8}$$

where we say that the adversary *forges* if it asks a decryption (second-oracle) query of $(N, A, \mathcal{C}) \in \mathcal{N} \times \mathcal{A} \times \mathcal{X}$ such that the oracle returns $\mathcal{D}_K^{N,A}(\mathcal{C}) \neq \textsc{Invalid}$ yet there was no prior query (N, A, P) that returned $\mathcal{C}$. We demand that $\mathcal{A}$ never ask two encryption (first-oracle) queries with the same first component (the N-value): it is nonce-respecting. We also demand that the adversary never asks a query outside of $\mathcal{N} \times \mathcal{A} \times \mathcal{X}$, that it never repeats a query, and that it never asks a decryption query of $(N, A, \mathcal{C})$ after having asked an encryption query (N, A, P) that returned $\mathcal{C}$.

The authenticity definition above "strengthens" the definitions in [178, 182] by allowing the adversary multiple chances to forge—every decryption attempt can be regarded as a forgery attempt. Informally, one expects security in the multiple-forgery-attempt definition to be q_{dec} times security in the single-forgery-attempt definition where q_{dec} is the number of decryption calls the adversary makes. The multiple-forgery-attempt definition is more consistent with other notions of security employed in this report, which is why I selected it.

III.3. Not bothering to define decryption. In describing GCM and CCM mode, we define encryption but don't bother with defining decryption. Our chosen syntax has made clear that, in general, defining the decryption function of an AE scheme is unnecessary in the sense that it is uniquely determined by the encryption function. Mathematically, decryption is whatever it "has to be" to be the inverse of encryption. We are not, however, suggesting that, in a spec, one shouldn't separately specify the decryption algorithm: it is certainly useful for those that will implement the scheme to *see* it, and its specification *is* important in addressing efficiency questions.

III.4. Short tags. Our definition of AE-security "gives up"—gives the adversary credit for "winning" in the experiment associated to defining $\mathbf{Adv}^{\mathrm{auth}}_{\mathcal{E}}(\mathcal{A})$—as soon as it achieves its first forgery. While this notion is perfectly good when we don't expect the adversary to ever forge—as is likely to be the case when tags are 64–128 bits—the definition shows its limitations if we want to consider tag-based AE with short tags, say tags of a single byte or, perhaps, tags of 32-bits. While there has been some recent academic work on MAC "reforgeability" and

dealing with short tags, especially Black and Cochran [35] and Wang, Feng, Lin, and Wu [197], we know of no definitional work addressing the problem for authenticated encryption. This limitation comes up in trying to understand and address Ferguson's criticism the GCM doesn't do well when tags are short not only because the first forgery is too easy but because, also, reforgeability is too easy [67]. We do not dispute this claim—we think it is true—but giving a provable-security-grounded discussion is made difficult because of the absence of a worked-out treatment.

Chapter 11

CCM Mode

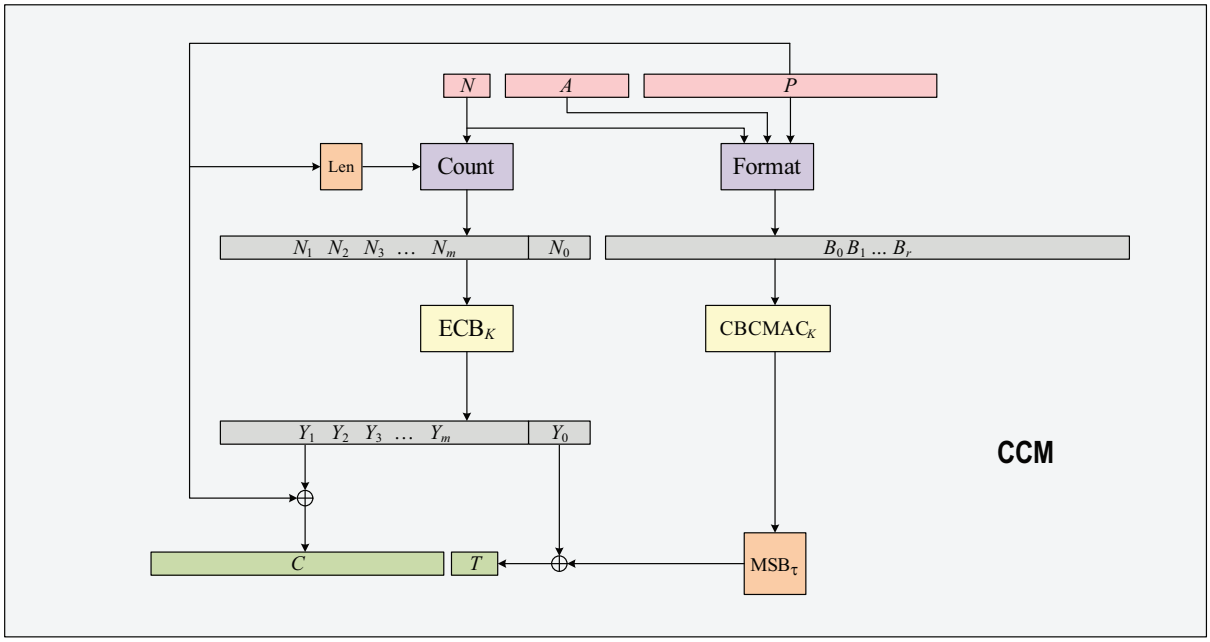
11.1. Summary. The CCM mode of operation—Counter with CBC-MAC—achieves AEAD (authenticated-encryption with associated-data) by combining CTR-mode encryption and the raw CBC-MAC. The amalgamation is in the style of MAC-then-Encrypt, but important changes make it differ from generic composition [23]. CCM was invented by Whiting, Housley, and Ferguson [203] and submitted to NIST. The initial motivation was to replace the OCB mechanism [182] that was in the draft IEEE 802.11i standard [87] by a patent-unencumbered scheme. While CCM was designed with that particular purpose in mind, it has, by virtue of its widespread standardization, become a general-purpose scheme.

Overall, CCM has many things in its favor. It enjoys a good provable-security guarantee. It has adequate performance for most applications. It is simple to implement—simpler than GCM. It is widely used, including its use in 802.11 (Wi-Fi networks), 802.15.4 (Low-Rate Wireless Personal Area Networks) (ZigBee), IPSec's ESP (RFC 4309), CMS (RFC 5084), and IKEv2 (RFC 5282), and RFC 3610. Critiques of CCM have pointed largely to efficiency issues (for example, the mode is not "on-line") and have not found anything damaging with respect to security [184]. Thus, despite deficiencies that I will detail, CCM probably should be included in any portfolio of important and provably-secure AEAD schemes.

In my analysis I re-express concerns about efficiency characteristics of CCM. I complain about the extremely low-level and *ad hoc* nature of its default "formatting function." I question NIST's decision to "isolate" the formatting and counter-generating functions to a non-binding appendix; as ungainly as these functions may be, I believe that their selection must, in the interest of interoperability and assurance, be considered as an intrinsic part of the standard, something that is fixed and mandated in any follow-on standardization. I also recommend that randomly-chosen IVs be explicitly disallowed (from my reading of the spec, this is somewhat ambiguous).

11.2. Definition of the mode. CCM is defined and illustrated in Figure 11.1. The mode is parameterized by a blockcipher $E\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ with an $n = 128$ bit blocksize, a tag length $\tau \in [32\,..\,128]$, a formatting function Format, and a counter-generation function Count. The functions Format and Count (which are not provided with any actual names in the spec) determine the message space, nonce space, associated-data space, and restrictions on the tag length.

One *particular* (Format, Count) pair is described in SP 800-38C; it is specified within [62, Appendix A]. I call it the *canonical* formatting and counter-generation functions and denote the pair of functions by (Format, Count). The functions are *themselves* parameterized by



10	**algorithm** $\text{CCM}_K^{N,A}(P)$	*CCM Encryption*
11	$m \leftarrow \lceil \lvert P \rvert / 128 \rceil$	
12	$N_0 N_1 \cdots N_m \leftarrow \text{Count}(N, m)$	
13	$Y_0 Y_1 \cdots Y_m \leftarrow \text{ECB}_K(N_0 N_1 \cdots N_m)$	
14	$C \leftarrow P \oplus Y_1 Y_2 \cdots Y_m$	
15	$B_0 B_1 \cdots B_r \leftarrow \text{Format}(N, A, P)$	
16	$Tag \leftarrow \text{CBCMAC}_K(B_0 B_1 \cdots B_r)$	
17	$T \leftarrow \text{MSB}_\tau(Tag) \oplus Y_0$	
18	**return** $C \parallel T$	
20	**algorithm** $\text{ECB}_K(X_0 \cdots X_m)$	
21	**for** $i \leftarrow 0$ **to** m **do** $Y_i \leftarrow E_K(X_i)$	
22	**return** $Y_0 Y_1 \cdots Y_m$	
30	**algorithm** $\text{CBCMAC}_K(X_0 \ldots X_m)$	
31	$Y \leftarrow 0^n$	
32	**for** $i \leftarrow 0$ **to** m **do** $Y \leftarrow E_K(Y \oplus X_i)$	
33	**return** Y	

Figure 11.1: **Definition of CCM.** The mode depends on blockcipher $E\colon \{0,1\}^k \times \{0,1\}^{128} \to \{0,1\}^{128}$, a tag length $\tau \in [32..128]$, a formatting function Format, and a counter-generation function Count. The formatting function determines the message space, nonce space, associated-data space, and the allowed tag lengths. At lines 14 and 17, the xor of unequal-length strings returns a string of the *shorter* length, ignoring the least-significant (rightmost) bits of the longer string.

a pair of numbers $t \in \{4, 6, 8, 10, 12, 14, 16\}$ and $q \in \{2, 3, 4, 5, 6, 7, 8\}$, which we might call the *byte length of the tag*, t, and the *byte length of the byte length of the message*, q. I write $\text{Format}_{q,t}(N, A, P)$ and $\text{Count}_t(N, m)$ to explicitly name the parameters (function Count does not depend on q). When $\text{Format}_{q,t}$ is used with CCM, the mode's message space is the set of strings having $2^{8q} - 1$ or fewer bytes, its nonce space is the set of strings having $(15 - q)$ bytes, and the tag length is $\tau = 8t$ bits. Under the canonical formatting and counter-generation function, the associated data must, independent of q and t, be a byte string of fewer than 2^{64} bytes.

Given the structure of the spec [62], with its Format- and Count-parameterized "core" and its canonically defined (Format, Count), one "should," in principle, separate discussion about CCM from discussion about CCM-with-(Format, Count). Still, to the best of my knowledge,

no formatting or counter-generation functions other than the canonical pair have ever been specified or used. In addition, all of the *other* standards specifying CCM, including RFC 3610 [204], fold the definition of the canonical formatting function and counter-generation function directly into the scheme that they too name CCM. In general, given the relatively complex requirements that the (Format, Count) pair must satisfy, given the complex engineering-tradeoffs implicit in selecting a definition for these objects, given the easy opportunity for error, and given that the goal of interoperability that is one of the aims of a cryptographic standard, I find it untenable to think that CCM is ever going to be used with anything other than the canonical formatting function and counter-generation function. I will, therefore, be somewhat cavalier in distinguishing CCM and CCM-with-(FORMAT, COUNT); in practice, the former is, it seems, the latter.

Following SP 800-38C [64], the underlying blockcipher for CCM *must* be AES. The requirement follows from assertions that the blockcipher must be a NIST-approved and have a 128-bit block size; only AES satisfies these requirements. (The only other NIST-approved blockciphers, Skipjack and TDEA, operate on 64-bit blocks.)

Quantity-of-use restrictions are placed on CCM, but they are quite liberal: the underlying blockcipher may be used for a total of no more than 2^{61} invocations. No special restrictions are placed on the use of "short" tags, in the spec defined as 64-bits or less, but there is a warning that the use of such tags needs to be be accompanied by a security analysis.

11.3. Canonical formatting function. Canonical formatting function $\text{FORMAT}_{t,q}(N, A, P)$ is complex; Appendix A of the NIST specification [62] comprised the most detailed and low-level pages I reviewed for this report. It might be noted, however, that the complexity is not the sort that would bother an implementer: realizing FORMAT in code is certainly simpler, for example, than realizing an efficient $\text{GF}(2^{128})$ multiply.

Function FORMAT is not named in the spec; this was my own attempt to rationalize what is going on. Here is the definition of $\text{FORMAT}_{q,t}(N, A, P)$, as previously extracted from the CCM specification in joint work with David Wagner [184].

$$
\begin{aligned}
&\text{FORMAT}_{q,t}(N, A, P) \;= && (11.1)\\
&\quad 0 \;\|\; \textbf{if } A = \varepsilon \textbf{ then } 0 \textbf{ else } 1 \textbf{ endif} \;\|\; [\, t/2 - 1 \,]_3 \;\|\; [\, q - 1 \,]_3 \;\|\; && (11.2)\\
&\quad N \;\|\; [\, |P|_8 \,]_{8q} \;\|\; && (11.3)\\
&\quad \textbf{if } A = \varepsilon \textbf{ then } \varepsilon \textbf{ elseif} && (11.4)\\
&\quad |A|_8 < 2^{16} - 2^8 \textbf{ then } [\, |A|_8 \,]_{16} && (11.5)\\
&\quad \textbf{elseif } |A|_8 < 2^{32} \textbf{ then } \texttt{0xFFFE} \,\|\, [\, |A|_8 \,]_{32} \textbf{ else } \texttt{0xFFFF} \,\|\, [\, |A|_8 \,]_{64} \textbf{ endif} \;\| && (11.6)\\
&\quad A \;\| && (11.7)\\
&\quad \textbf{if } A = \varepsilon \textbf{ then } \varepsilon \textbf{ elseif } |A|_8 < 2^{16} - 2^8 \textbf{ then } \texttt{(0x00)}^{(14-|A|_8) \bmod 16} && (11.8)\\
&\quad \textbf{elseif } |A|_8 < 2^{32} \textbf{ then } \texttt{(0x00)}^{(10-|A|_8) \bmod 16} \textbf{ else } \texttt{(0x00)}^{(6-|A|_8) \bmod 16} \textbf{ endif} \;\| && (11.9)\\
&\quad P \;\| && (11.10)\\
&\quad \texttt{(0x00)}^{(-|M|_8) \bmod 16} && (11.11)
\end{aligned}
$$

Above, the nonce $N \in \{0,1\}^{8(15-q)}$ must comprise $15-q$ bytes. Note that the string B returned as $\text{FORMAT}_{q,t}(N, A, P)$ will always have a positive multiple of 128 bits.

The complexity underlying FORMAT is clearly meant to promote concision; the inventors want that the value B returned formatting N, A, and P should not be unnecessarily bigger than the sum of the length of those arguments. The encoding is also required to have some

particular properties, relative to COUNT, explicitly named in [62, Section 5.4] and coming out of the proof of Jonsson [103]. I will return later to those properties.

11.4. Canonical count-generation function. The function $\textsc{Count}_q(N, m)$ is comparatively trivial; we define $\textsc{Count}_q(N, m)$ as $N_1 \parallel N_2 \parallel \cdots \parallel N_m$ where

$$N_i \;=\; 0^5 \;\parallel\; [\,q-1\,]_3 \;\parallel\; N \;\parallel\; [\,i\,]_{8q} \tag{11.12}$$

Recall that N is $15-q$ bytes and each N_i is a 16-byte block.

11.5. Provable-security results. The CCM mode of operation enjoys good provable-security results due to Jakob Jonsson [103]. Let's begin with privacy. Jonsson employs the privacy and authenticity definitions of §III.2. For privacy, a nonce-respecting adversary $\mathcal{A}$ makes a sequence of (N, A, P) calls, trying to tell if it is speaking to a CCM_K oracle or an oracle that returns the same number of random bits. During the attack, let

- σ bound the total number of blockcipher calls that the mode would make on the sequence of queries, following the CCM specification.

Then Jonsson [103, Theorem 2] proves that

$$\mathbf{Adv}^{\text{priv}}_{\text{CCM}[\text{Perm}(n),\tau]}(\mathcal{A}) \;\le\; \frac{\sigma^2}{2^n}\,. \tag{11.13}$$

Here we write $\text{CCM}[\text{Perm}(n), \tau]$ for the CCM scheme as it would be instantiated with an ideal n-bit blockcipher (that is, E_K is a uniformly random permutation) and tag-length τ. The result holds for any (Format, Count) functions satisfying specified properties. In particular, it holds for $(\textsc{Format}_{q,\tau/8}, \textsc{Count}_q)$.

Passing to the complexity-theoretic setting is easy and standard; when a "real" blockcipher E is used, the privacy advantage increases by an amount that is at most $\mathbf{Adv}^{\text{prp}}_E(\mathcal{B})$ for an adversary $\mathcal{B}$ that asks σ oracle queries to its blockcipher, the adversary running in essentially the same amount of time as did $\mathcal{A}$.

Since we require $n = 128$ in CCM, formula (11.13) provides a good privacy bound up to nearly 2^{64} queries. In particular, given that the standard mandates use of the PRP for no more than $\sigma = 2^{61}$ blockcipher calls, an adversary's privacy advantage can never exceed 0.016 plus the PRP-insecurity of the underlying blockcipher (AES)—with far smaller information-theoretic offset for more realistic values of σ.

For authenticity, recall that the adversary asks a series of encryption or decryption queries, the former nonce-respecting, and its aim is to ask a decryption query that's a *forgery*: it returns something other than INVALID even though this is *not* an (N, A, C) query that follows an earlier (N, A, M) query that returned C. As the adversary $\mathcal{A}$ caries out its attack, let

- q_{dec} bound the total number of decryption queries that it asks, and let
- σ bound the total number of blockcipher calls that the mode would make on the adversary's sequence of queries, both to encrypt all the encryption queries and to decrypt all the decryption ones.

Then Jonsson [103, Theorem 1] shows that

$$\mathbf{Adv}^{\text{auth}}_{\text{CCM}[\text{Perm}(n),\tau]}(\mathcal{A}) \;\le\; \frac{q_{\text{dec}}}{2^\tau} + \frac{\sigma^2}{2^n}\,. \tag{11.14}$$

This is a good bound: the probability of forgery grows linearly in $2^{-\tau}$, which is unavoidable, with an additional term that stays tiny until $\sigma \sim 2^{64}$. As an example, if we truncate tags to 32-bits, and let the adversary ask at most $10,000$ forgery attempts before the key must be changed or the connection dropped, and if the total amount of plaintext or ciphertext that the adversary attempts to encrypt or decrypt corresponds to 2^{50} blocks, then the adversary's probability of forging will be at most $10000/2^{32} + 2^{-28} \approx 10000/2^{32} < 0.000003$.

For the complexity-theoretic version one simply adds the usual $\mathbf{Adv}^{\mathrm{prp}}_{E}(\mathcal{B})$ term.

As we will see in Chapter 12, the authenticity bound of CCM, equation (11.14), stands in sharp contrast to the authenticity bound for GCM, equation (12.2), where security falls off in $\sigma^2/2^{\tau}$ instead of $\sigma^2/2^n$. What is more, the difference is actually due to an attack on GCM. The upshot is that CCM is a perfectly acceptable AEAD mechanism to use with truncated tags—and the standard quite reasonably permits truncation to as few as 32-bits. In contrast, GCM cannot be used with such a tag length without making unreasonably restrictive assumptions as to amount of material that the scheme will act upon.

The privacy bound of CCM is tight, up to a constant; there are simple attacks to show this. It is not known if the authenticity bound is tight, a problem that Jonsson himself points to as being of interest [103, Problem 1]. Progress on this question was made by Fouque, Martinet, Valette, and Zimmer [71]. They show that a two-independent-key variant of CCM has a better authenticity bound than (11.14)—the second addend is reduced, at least in some cases—*if* one measures security of the underlying blockcipher with respect to the PRF-measure instead of the PRP measure. In other words, they assume a random function instead of a random permutation in carrying out the analysis, and sometimes get a better bound. Still, the improvement is minor, still showing birthday-bound behavior—or worse—in the worst case. (Here I am referring to Theorem 4 of [71] having the term $256q_v((s+1)^2/2^n)^2$, which can be even worse than $\sigma^2/2^n$.) Besides the small improvement, it is not clear that the result has any significance with respect to the "real" CCM, where there is one key and one starts from a good PRP rather than a good PRF. In conclusion, we do not know if the authenticity bound for CCM is tight, but in some sense it does not matter, as the existing bound is already quite good.

11.6. Efficiency issues. I discuss three efficiency problems with CCM: (a) CCM is not on-line, (b) CCM disrupts word-alignment, and (c) CCM can't pre-process static associated data. These comments are adapted from an earlier note I prepared with David Wagner [184].

11.6.1 *CCM is not on-line.* An algorithm is *on-line* if it is able to process a stream of data as it arrives, with constant (or at most logarithmic) memory, not knowing in advance when the stream will end. Observe then that on-line methods should not require knowledge of the length of a message until the message is finished. CCM fails to be on-line in both the plaintext and the associated data: one needs to know the length of both before one can proceed with encryption or decryption.

It is true that, in many contexts where one is encrypting or decrypting a string, one *does* know its length from the beginning of the process. But there are also contexts where one does *not* know the length of the message in advance of getting an indication that it is over. In inability to handle on-line message arrivals makes it impossible to support a good incremental API.

11.6.2 *CCM disrupts word-alignment.* Most modern machines perform operations much more efficiently when pointers into memory fall along word-boundaries (which typically occur every 4 or 8 bytes). A typical software implementation of a CBC MAC, for example, will exhibit much worse performance if it is called on an argument that is not word-

aligned. The canonical formatting function FORMAT does not ensure that, as we MAC the formatted string B, the constituent part for A will be aligned on a word boundary. If it is not, expect the performance to suffer.

11.6.3 *Can't pre-process static AD.* In many scenarios the associated data A will be static over the course of a communications session. For example, the associated data may include information such as the IP address of the sender, the receiver, and fixed cryptographic parameters associated to this session. In such a case one would like that the amount of time to encrypt or decrypt a string should be independent of $|A|$ (disregarding the work done in a preprocessing step). The reason that CCM fails to allow pre-processing of associated data is that the canonical formatting function encodes the nonce N and the message length $|P|/8$ before A rather than after it.

I do not suggest that any of these performance issues is devastating. What I do suggest is that all of these problems *can* be significant, in some settings. What is more, none of these problems were necessary—all of them could have been avoided by alternative designs.

11.7. Paramaterization. The user of CCM (assume, as usual, its canonical formatting function) must select a parameter "q", the byte length of the byte length of the longest message. The user has no business seeing or selecting such a low-level parameter. While it is reasonable and customary to fix a suitably large maximum message length, such as $2^{64} - 1$ bytes, it seems unreasonable to force the user to think about choosing small values, like $q = 2$, or to understand the CCM details that are needed in order to select the value q. Worse still, the definition of CCM involves a tradeoff between two things that are conceptually unrelated: the maximal message length and the length of the scheme's nonce. Bizarrely, from a user's point of view, the nonce length will be defined as $15 - q$. What, conceptually, is the relationship between the nonce length and the length of longest plaintext one may encrypt or decrypt? There is none. It is just an artifact of trying to shove a nonce and a length encoding into 15 bytes.

11.8. Complexity. While human-perceived complexity is inherently subjective, I find CCM—particularly its canonical encoding function—to be quite complex.

A glimpse of CCM's complexity can be seen from the fact that correctness crucially depends on the encoding convention. For example, the spec disallows the possibility of $t = 2$ (two-byte tags), which means that at least one of bits 3, 4, or 5 of the first block of B, which holds $[\, t/2 - 1 \,]_3$ is non-zero, while these bits are always zero in the initial counter value. If one *had* allowed a tag of $t = 2$ bytes, the scheme would be utterly insecure. In short, CCM's correctness is integrally wrapped up in encoding-scheme details.

As another way to evidence CCM's complexity, ask the question: *how many block cipher calls does CCM use?* For most modes—indeed for *all* of the other modes looked at within this report—one can write down a simple and understandable formula. For CCM, the answer is

$$\mathit{NumCalls}(A, P) \;=\; 2\left\lceil \frac{|P|}{128} \right\rceil + \left\lceil \frac{|A|}{128} \right\rceil + 2 + \delta(|A|)$$

where $\delta(a) \in \{0, 1\}$ is defined by letting

$$\lambda(a) \;=\; \begin{cases} 0 & \text{if } a = 0 \\ 16 & \text{if } a \in [8, 8 \cdot (2^{16} - 2^8 - 1)] \\ 48 & \text{if } a \in [8 \cdot (2^{16} - 2^8), 2^{35} - 8] \\ 80 & \text{if } a \in [2^{35}, 2^{64} - 8] \end{cases}$$

and setting $\delta(a) = 0$ if $(a \bmod 128) + \lambda(a) \leq 128$, and $\delta(a) = 1$ otherwise.[1]

In reading the spec, the obvious contributor to complexity is all of the "bit twiddling" that CCM does within its formatting function FORMAT. While I find this bit-twiddling ungainly, this is ultimately a matter of taste, and I am sympathetic to the argument that this is a "shallow" sort of complexity. Fouque *et al.* goes so far as to assert that "[t]he security of CCM is very interesting since it relies on some padding or formatting functions. Such requirements are not appreciated in general and cryptographers try to avoid such properties" [71, p. 412].

A more fundamental kind of complexity in CCM stems from the absence of a clean abstraction boundary within the scheme: the mode is not an instance of generic composition [23], and it is not designed to use any VIL-secure message authentication code. Even though the scheme is "based" on CTR mode, I myself found it overly awkward to even express the pseudocode that way: in Figure 11.1 I expressed the scheme in terms of ECB.

Ferguson points out that, for all CCM's apparently complexity, "it takes only a handful lines of code to perform the CCM encoding. That should be compared to hundreds of lines of code for the full CCM mode (excluding the AES), so the 'extra' complexity is actually quite small. Something like GCM is hugely more complicated to implement, because of the GF(2^n) arithmetic" [68]. Ferguson goes on to suggest that there would seem to be "a tradeoff between engineering simplicity and mathematical simplicity. And given that a successful mode will be analyzed by dozens of mathematicians and many hundreds of engineers, I still maintain that engineering simplicity is preferable over mathematical simplicity" [68]. There is truth, I think, to all of these comments.

There seems to be little that CCM achieves that a generic composition of CTR and CMAC could not also have achieved—more simply and, probably, with greater efficiency. Concretely, an Encrypt-then-MAC amalgamation of CTR mode and CMAC would make for a simple and efficient scheme, albeit one with two separate keys. If one were concerned about that issue, it alone could have been the target of violating the abstraction boundary. I note that the implicit "proposal" of this paragraph ignores the historical reality that CCM was standardized by NIST a full year prior to CMAC's standardization [62, 63].

11.9. Random nonces? There are some problems with the NIST document itself [62] that I should mention. To begin, I simply cannot determine if nonces may be random. On my first reading, the answer seemed clearly to be *no*. The spec asserts that a nonce is a *value that is used only once within a specified context* [62, p. 4]. Later, the spec says again that *[the] nonce shall be non-repeating* [62, p. 8]. There are absolute statements, not probabilistic ones, so it seemed clear that random nonces could *not* be used. This seemed further supported by common sense, as nonces under CCM may be as short as seven bytes, which is too small for safely using a random value. Yet immediately following the second quote is an assertion that *[the] nonce is not required to be random*—which certainly *seems* suggest that the nonce *may* be random. It should be clarified if nonces may or may not be random. Given that CCM be used with quite short nonces, and given the difficulty of verifying that "random" nonces actually are, I would suggest that random nonces should not be used with CCM.

Niels Ferguson, co-inventor of CCM, concurs with the interpretation that nonce-requirement is absolute; a uniformly random nonces is not allowed. He interprets *[the] nonce is not required to be random* [62, p. 8] to mean only that the nonce need not to be unpredictable—a counter, for example, is just fine [68].

[1] As per communication from Niels Ferguson, there would seem to be an error in the definition just given for $\delta(a)$. The exact formula is not really important; the point is only that it is quite complex.

11.10. Decryption checks and timing attacks. There would seem to be another issue in the spec of the algorithm in the vague assertion of Step 7 in the "Decryption-Verification Process" [62, Section 6.2], which instructs that "If N, A, or P is not valid, as discussed in Section 5.4, then return INVALID, else apply the formatting function to (N, A, P) to produce $B_0, B_1, \ldots, B_r$." No INVALID-returning event is specifically described in Section 5.4 of the spec, but apparently this has to do with the generation of domain points outside of those allowed by the formatting function. Regardless, there is no corresponding checks in the Generation-Encryption Process, and the two processes *must* match in terms of any such checks. Finally, the spec asserts that an "implementation shall ensure that an unauthorized party cannot distinguish whether the error message results from Step 7 or from Step 10, for example, from the timing of the error message" [62, p. 11]. This is wrong at multiple levels. First, it is not clear what other "examples" are in scope of this "for example." More fundamentally, it is strange that a functional spec—that *any* function-centered spec—should suddenly turn towards describing an *implementation* requirement to resist side-channel analyses. Not only may such attacks be irrelevant in many contexts where CCM might be used, but if one *is* going to try to resist side-channel attacks, it is going to require a discussion that extends far beyond a 32-word sentence that seems to have dropped, without foundation, into the spec.

All that said, CCM is a mode that probably *can* be implemented efficiently in a manner that resists timing attacks, assuming the underlying AES implementation is so implemented as well. This is one of its advantages over a table-based GCM.

11.11. Software performance. The inherently sequential nature of CBC mode, and the need for two passes with AES, can make CCM-AES a slower scheme, in software, then GCM or other AEAD modes. That said, the difference are often not large, and are often not relevant. I defer a discussion to §12.7.

11.12. Concluding remarks. While I have, as usual, been quite critical, we should keep the following in perspective. First, CCM is already widely used. Second, CCM achieves good provable-security results, even with truncated tags. While it is not a terribly fast or pretty mode, it does the job, and it's been doing the job for some time. The most "important" suggestion, in the end, may be to mandate use of the canonical formatting and counter-generation functions. It would also be good to clarify if random nonces are or are not allowed.

Chapter 12

Galois/Counter Mode

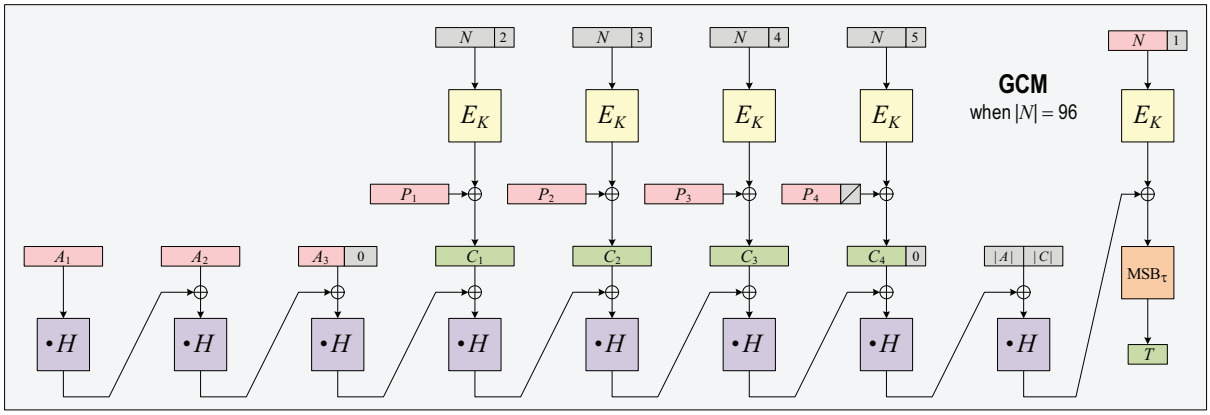
12.1. Summary. Galois/Counter Mode (GCM) achieves AEAD (authenticated-encryption with associated-data) by combining CTR-mode encryption and Carter-Wegman message authentication [50, 201]. The amalgamation is in the style of encrypt-then-MAC, but important differences make it differ from generic composition [23]. The universal hashing underlying the Carter-Wegman authentication is based on polynomial evaluation over $\mathrm{GF}(2^{128})$, a classical construction rooted in folklore.

GCM was invented by David McGrew and John Viega [133, 137]. It was offered as an improvement to the similarly-conceived CWC mode of Kohno, Viega, and Whiting [115], whose universal hashing was based on integer rather than finite-field multiplication. The change particularly benefits hardware embodiments. GCM is standardized by NIST as SP 800-38D [64], on which this evaluation is based.

Overall, GCM has many things in its favor. It has a number of desirable performance characteristics, soon to be detailed. It enjoys a good provable-security guarantee, assuming that its tags are left long. It is becoming widely used, including its use in IPsec, MACSec, P1619.1, and TLS, and [88, 89, 136, 186, 195]. The mode can be quite efficient in both hardware and software, and direct hardware support is now being offered on new Intel microprocessors [79]. Critiques of GCM, particularly Ferguson's [67], raise significant issues, but have not been fatal. Thus, despite deficiencies I will detail, GCM ought to be included in any portfolio of important and provably-secure AEAD schemes.

In my analysis I express concerns about the use of GCM with short tags (anything fewer than 96 bits); I recommend disallowing this. I also favor mandating or otherwise encouraging the use of 96-bit IVs, the use of both shorter and longer tags increasing conceptual complexity, implementation complexity, and proof complexity. For table-based implementations of GCM, timing attacks cannot be discounted as a real possibility. Finally, there are some issues with NIST's spec, which would have done well to confine its body to defining the algorithm, isolating further information to a (better written) informational annex.

12.2. Definition of the mode. GCM is defined and illustrated in Figure 12.1. The mode is parameterized by a 128-bit blockcipher $E\colon \{0,1\}^k \times \{0,1\}^{128} \to \{0,1\}^{128}$ and a tag length $\tau \in \{32, 64, 96, 104, 112, 120, 128\}$. Following SP 800-38D [64], the underlying blockcipher *must* be AES (with any of its three key lengths). The requirement follows from assertions that the blockcipher must be a NIST-approved and have a 128-bit block size; only AES satisfies these requirements. (The only other NIST-approved blockciphers, Skipjack and TDEA, operate on 64-bit blocks.)




```
10  algorithm GCM_K^{N,A}(P)                                          GCM Encryption
11  // |P| ≤ 2^39 − 256,  |A| < 2^64,  |N| > 0,  |N| < 2^64,  8 | |P|,  8 | |A|,  8 | |N|
12  H ← E_K(0^128)
13  if |N| = 96 then Cnt ← N ‖ 0^31 1
14              else Cnt ← GHASH_K(N ‖ 0^i ‖ |N|_128) for minimal i ≥ 0 s.t. 128 | (|N| + i)
15  m ← ⌈|P|/128⌉;   for i ← 0 to m do Y_i ← E_K(Cnt + i)
16  C ← P ⊕ (Y_2 ‖ Y_3 ‖ Y_4 ‖ ···)
17  X ← A ‖ 0^i ‖ C ‖ 0^j ‖ |A|_64 ‖ |C|_64 for minimal i, j ≥ 0 s.t. 128 | (|A| + i) and 128 | (|C| + j)
18  Tag ← Y_1 ⊕ GHASH_H(X)
19  T ← MSB_τ(Tag)
20  return C ‖ T

30  algorithm GHASH_H(X)                                              Used internally
31  X_1 ··· X_m ← X where |X_i| = 128
32  Y ← 0^128;   for i ← 1 to m do Y ← (Y ⊕ X_i) • H
33  return Y
```

Figure 12.1: **Definition of GCM.** The mode depends on blockcipher $E\colon \{0,1\}^k \times \{0,1\}^{128} \to \{0,1\}^{128}$ and tag length $\tau \in \{32, 64, 96, 104, 112, 120, 128\}$. At line 15, addition of string *Cnt* and integer i adds the number into the last 32-bits of the string, ignoring any carry. At line 16, the xor of unequal-length strings returns a string of the *shorter* length, ignoring the least-significant bits of the longer string.

Quantity-of-use restrictions are placed on GCM if using either of the smallest two permitted tag lengths [64, Appendix C]. Specifically, for 32-bit tags, an implementation must ensure that when the maximal permitted message length is 2^5 (or 2^6, 2^7, 2^8, 2^9, or 2^{10}) bytes, the maximal permitted number of applications of the decryption function does not exceed 2^{22} (or 2^{20}, 2^{18}, 2^{15}, 2^{13}, 2^{11}, respectively). Limits are not specified if the maximal permitted message length exceeds 2^{10} bytes; one would guess that the intent was that such messages not be allowed. Similarly, for 64-bit tags, if the maximal permitted message length is 2^{15} (or 2^{17}, 2^{19}, 2^{21}, 2^{23}, or 2^{25}) bytes, the maximal permitted number of applications of the decryption function must be no more than 2^{32} (or 2^{29}, 2^{26}, 2^{23}, 2^{20}, 2^{17}, respectively). Limits are not specified if the maximal permitted message length exceeds 2^{25} byte;. one would guess that the intent was that such messages not be allowed.

12.3. Desirable characteristics. Among GCM's prominent and generally desirable characteristics, I would enumerate the following:

12.3.1 Assuming tag lengths equal to the blocklength, GCM enjoys provable security up to the usual birthday bound [133]. The underlying cryptographic assumption is the standard

one, that the blockcipher is a good PRP. See §12.5 for further discussion on McGrew and Viega's bounds.

12.3.2 Ciphertext expansion in GCM is minimal: when encrypting a plaintext one gets back a ciphertext having identical length to that of the plaintext, plus an authentication tag of 4–16 bytes.

12.3.3 All that is expected of the IV is that it be as a nonce (each value used at most once in a given session); it is not required to be random or unpredictable. See §III.2 for the notion of nonce-based AEAD. To emphasize the minimal expectations on the IV, we routinely refer to it as a nonce, N, rather than as an IV.

12.3.4 Only the forward direction of the blockcipher is used in the mode. This saves chip area compared to AEAD constructions (eg, OCB [179, 182]) that require the backwards direction of the blockcipher for AEAD-decryption.

12.3.5 GCM is fully parallelizable, enabling hardware throughput that is not limited by blockcipher-latency, and benefiting software embodiments as well. At high data rates, stalls introduced by a mode of operation reduce the rate at which the implementation can process data. There is a significant opportunity cost if a mode needs to stall before some invocations of the blockcipher.

12.3.6 On decryption, the authenticity of the protected data can be verified independently from the recovery of the confidential data, a fact that can have desirable architectural or performance implications. For example, invalid messages can be ignored without CTR-mode decrypting them.

12.3.7 GCM is efficient for hardware, since the CTR mode portion of GCM is simple and it is efficient, in hardware, to construct a $\mathrm{GF}(2^{128})$ multiplier, assuming multiplication is broken down over several (eg, four or eight) clocks. (On the other hand, if one wanted a single combinatorial circuit that did the entire $\mathrm{GF}(2^{128})$ multiply at once, it would not be so small.) See Zhou, Michalik, and Hinsenkamp for recent FPGA hardware estimates [208]. Overall, in hardware, GCM is unsurpassed by any authenticated-encryption scheme.

12.3.8 GCM can be quite efficient for software as well. I will return to a discussion of software performance in §12.7, as the story here is somewhat nuanced.

12.3.9 Unlike CCM (Chapter 11), GCM is on-line: one does not need to know the length of the message (nor the length of the AD) in advance of processing it. (One does, however, need to know the AD and its length before processing the plaintext.) This makes the mode suitable for some streaming and networking applications where CCM cannot be used. It also enables an incremental API (Application Programmers Interface), where the plaintext, ciphertext, or associated data is provided incrementally in chunks.

12.3.10 If the AD is static (it changes rarely or not at all across a session), then it can be pre-processed so that there is effectively no per-message cost to providing the AD's authenticity.

12.3.11 If the nonce is predictable and the message length is known or reasonably bounded, the AES-enciphering operations can also be precomputed. After subkey-generation, AES is used in the mode exclusively for generating the pad $E_K(N)\, E_K(N+1)\, E_K(N+2) \cdots$.

12.3.12 Like any AEAD mode, GCM can be used for authenticity (no encryption) as a nonce-based MAC. For this setting the MAC has been given a separate name, GMAC, and is

regarded by SP 800-38D as a distinct mode of operation. See Chapter 10.

12.3.13 When used for message authentication, GCM in incremental, in the sense of Bellare, Goldreich, and Goldwasser [15], with respect to appending and substituting 128-bit blocks.

12.3.14 GCM is a permitted option for IPsec's ESP and AH, and for TLS [136, 186, 195]. GCM author McGrew has been active in supporting GCM's standardization and use, authoring multiple related IETF RFCs (namely, RFC 4106, 4543, 5116, 5282, 5288, and 6054). GCM is effectively "guaranteed" to be well-aligned with McGrew's higher-level AEAD proposed standard RFC 5116 [131].

12.3.15 Beginning with Intel "Westmere" processors, the company is producing microprocessors that natively support not only AES computations but also what they call carryless multiply, PCLMULDQ. The NI (new instruction) is aimed at GCM. Its use, along with the AES NIs, greatly speeds up GCM computation [79].

12.3.16 The GCM algorithm is not itself covered by any known patents (but see 12.4.11 for some pending and granted patents that have emerged).

Many of the advantages named above are identified in McGrew and Viega's papers on GCM [133, 137] and in the NIST standard itself [64].

12.4. Undesirable characteristics. I now list some unpleasant characteristics of GCM and the NIST Recommendation in which it is embedded.

12.4.1 When GCM tags are truncated to τ bits, Ferguson describes a message-forgery attack that uses an about 2^τ (n-bit) blocks to succeed [67]. The adversary asks for the MAC of a single message having $2^{\tau/2+1}$ blocks and then forges after about $2^{\tau/2}$ expected tries, on messages again of $2^{\tau/2+1}$ blocks. Focusing on the case of 32-bit tags, for example, the attack begins by asking for the MAC of a single message of 2^{17} blocks. After that, it needs about 2^{16} verification messages, again of 2^{17} blocks each, until the first forgery is expected. After that, a comparable number of further queries leads to a complete authentication failure—recover of the key H. For eight-bit tags (something that is *not* allowed by the standard) the problem would be even worse; here, Ferguson estimates that an attacker can recover the subkey H with about 1000 forgery attempts, each of 48 bytes [67, Section 5.1].

There are actually two halves of the problem. One is that the first forgery comes too quickly—not after something close to 2^τ forgery attempts (of $\sigma = 2^{n/2}$ blocks of messages), as one might expect, but after $2^{\tau/2}$ forgery attempts (and 2^τ blocks of messages). The second problem is that, once the first forgery happens, additional forgeries happen too fast. After the first failure, all authenticity is completely forfeit.

The attack described is not a concern if long tags are used—say 96–128 bit tags—but it is a problem if short tags are used. Correspondingly, the NIST spec mandates rather aggressive re-keying for 32-bit and 64-bit tags [64, Appendix C]. The requirements are fairly onerous (see §12.2) and their realization could be awkward or even infeasible.[1]

I do not "take sides" between Ferguson [67] and McGrew and Viega [135] as to how common short tags may be for contexts that might like to use GCM. I do not know. But I side with Ferguson in believing that truncating tags to τ bits *ought* to deliver

[1] Under a typical API, an encryption subsystem will typically not *know* what is the maximal length that a combined ciphertext and AD may have. Nor do systems routinely keep track of the number of blocks that have been decrypted. Nor will the aggregation of such information necessarily be architecturally feasible.

forgery probabilities of about $2^{-\tau}$ plus the insecurity of the underlying blockcipher. A bit more precisely, the expected number of verification queries until the first forgery should be about $2^{\tau-1}$ plus an error term that grows with something that is quadratic in the adversary's communications resources divided by 2^n. Not something that is quadratic in the adversary's communications resources divided by 2^τ.

I disagree with McGrew and Viega [135] that it is "unfair" to criticize GCM for failing to deliver forgery probabilities close to $2^{-\tau}$ [135, p. 2]. The authors cite one of my own paper [19] to explain that forging probability is *expected* to fall off with parameters like the number of queries and their lengths. That much is true, but they fail to explain that, in that very example they cite, the decline in security tracks something inverse exponential in the blocklength, not something inverse-exponential in the tag length. The difference is important.

It is worth nothing that 32-bit MACs have been the norm in retail banking for decades [3]. When used in a construction like the "retail MAC" (§7.13), forgeries with query-complexity near $\sim 2^\tau$ blocks are not known. In fact, they demonstrably cannot arise if the underlying blockcipher is a good PRP; our provable-security bounds are better than that.

As suggested above then, the relative ease of forging a short tag with GCM is not a necessary feature of MACs or AEAD schemes. The problem cannot arise if one authenticates a message with a good PRF (rather than something that is merely a good MAC). Nor can the problem arise in the AEAD setting if one accomplishes authentication with the encrypt-then-PRF paradigm [23] and one is using a good PRF. In both settings one will get authenticity bounds that are approximately the insecurity of the PRF (which is not weakened by truncation and never depends on τ) plus $2^{-\tau}$.

The ease-of-first-forgery issue does arise if one truncates the raw CBC-MAC (§7.11) in an attempt to turn it into a VIL PRF. This is the substance of Lars Knudsen's nice attack [113]. Note that there is no contradiction between the existence of this attack and the fact that the CBC-MAC is a provably-good PRF (and good PRFs never exhibit first-forgery degenerate behavior when truncating to make a MAC) since the raw CBC-MAC is only secure in the FIL setting, and there is no guarantee that truncation of an FIL-secure PRF will yield a VIL-secure MAC of *any* particular quality.

One also sees the short-tag problem arise if one truncates a conventional Carter-Wegman MAC (which is what happened with GCM). Yet the problem can be avoided relatively easily [117, 197]. The papers just cited provide a clear discussion of the forgery issue for the truncated-tag message-authentication setting.

No papers in the literature that have specifically taken up this tag-truncation problem in the setting of AEAD, and there is nothing to generically guarantee that a "good" AEAD scheme remains good when its tags are truncated. There are simple counterexamples to show that no such result is possible under our definitions. Nor do our definitions of AEAD deal with the ease-of-additional-forgeries issue; the AEAD notion is too coarse for that, effectively "giving up" when the first forgery happens, declaring the adversary victorious. See Black and Cochran [35] for a provable-security treatment of MAC reforgeabilty, and see Handschuh and Preneel[85], following McGrew and Fluhrer [132], for a more attack-centric view of the problem.

Summarizing, the ease-of-first-forgery issue shows up in the bounds; the definitions are fine. The ease of re-forgery issue lies outside of our notions used for AEAD. I personally feel a share of the responsibility for this; it is my own definition of AEAD

that fails to deal well with very-short tags. This could be regarded as a "defect" in the definitions; one might like to have a sensible definition of AEAD security that makes sense even if tags are, say, a single bit.

See §12.5 and §12.6 for further discussion on this topic.

12.4.2 Closely tied to short-tag problem is the fact that the provable security bounds for GCM are disappointing, showing a degradation in authenticity (and also privacy) that falls off with $qm2^{-\tau}$ where q is the number of queries and m bounds a longest message (measured in blocks and including the AD's length, too). Of course the poor bound is better than a wrong one, and, as McGrew and Viega correctly point out [135], all that Ferguson's attack ultimately does is to establish that their bound is tight. Still, one would prefer to have AEAD schemes with authenticity bounds that fall off in $q_v 2^{-\tau}$ (plus additional terms not depending on τ), where $q_v < q$ is the number of verification queries. See §12.5.

12.4.3 While GCM may seem similar to Bellare and Namprempre's favored encrypt-then-MAC approach to generic composition [23], with the MAC realized as a PRF, it is not an encrypt-then-PRF scheme. The most obvious difference is the non-independence of the keys K and H. More subtle and more important, however, is that the tag is not computed by applying a PRF (a nonce-dependent Carter-Wegman MAC is not one). Nor is the MAC even taken over the "right" fields, which, in the nonce-based setting, should have included the nonce [180]. Using a true encrypt-then-PRF approach might have gone a good ways to conceptually simplifying the scheme, and would have resulted in something without the short-tag vulnerability as well. That said, it is true that there would have been a cost. For example, one could not have claimed efficiency 12.4.11 if one had to encipher a computed hash to make the PRF, including or not including the nonce (see [35] and options 2 and 3 of [85, p. 150]).

12.4.4 Having the field point $a_{127}u^{127} + \cdots + a_1 u + a_0$ correspond to the 128-bit string $a_0 a_1 \cdots a_{127}$ [64, p. 11] instead of $a_{127}a_{126} \cdots a_1 a_0$ was not a good idea.[2] The GCM inventors call their choice the "little endian" convention [134, p. 20], as does the NIST Recommendation [64, p. 12], but the convention really has nothing to do with the customary meaning of endianness, which speaks to the ordering of bytes in external memory and how they are mapped into the machine's registers on LOADs and STOREs. When instead speaking of the arrangement of bits within a register (or whatever logical units an ALU is operating on), big-endian and little-endian machines do exactly the same thing: *most-significant-bits come first.* Indeed the convention is so entrenched that Intel declined to use GCM's reflected-bit representation (my term) for defining carryless multiplication (PCLMULDQ) *despite* this instruction being targeted for GCM. Using the reflected-bit convention would have resulted in a "unique" representation of numbers. As a result of the mismatched conventions, a GCM implementation based on the PCLMULDQ instruction must go through clever gyrations to match GCM's conventions without sacrificing too much speed [79].

Another consequence of the reflected-bit convention is an inability to use addition to emulate "doubling," here meaning multiplication by u. With the GCM convention, doubling corresponds to a (logical) right-shift instead of a left-shift. On existing processors, a right-shift by one is never, to the best of my knowledge, faster than a left-shift by one accomplished by way of the common idiom $R_i \leftarrow R_i + R_i$. It can be slower.

[2] Some of the observations made in this item are due to Ted Krovetz (2010). Many thanks, Ted, for explaining these points to me.

McGrew and Viega do offer an explanation for their convention. "We chose to use a 'little endian' definition of the field," they write, because it "allows a multiplier to process data as it arrives ... whenever the width of the data bus is less than 128 bits." I doubt this comes up very often at all, and suspect this is a case of "over-optimization," subverting common practice for marginal and questionable gain.

Note that the reflected-bit convention makes the representation of field points differ across the NIST standards for CMAC [63] and GCM [64]. It is unfortunately that NIST flipped representation conventions for $\mathrm{GF}(2^{128})$ after already making an initial choice. But the choice had already been embedded in prior standards.

12.4.5 GCM plaintexts are limited to about 68.7 Gigabytes ($2^{36} - 32$ bytes). While the limit is of debatable practical importance, it is, at the very least, unaesthetic and artifactual. It arises out of the need to ensure that, when $|N| = 96$, we never increment the four-byte block counter that is appended to the nonce until it would wrap to 0^{32}.

12.4.6 GCM works very differently when $|N|$ is or is not 96 bits. Not only is the mechanism less "natural" for the $|N| \neq 96$ setting, but security is harder to see and to prove. And since nonce-lengths can vary in a session (tag lengths are fixed for a context, but nonce lengths are not) the proof must carefully attend to demonstrate the lack of "interaction" between the two ways of dealing with the nonce N. While such an analysis *is* provided by McGrew and Viega's proof [133, 137], tricky proofs are infrequently verified, while simplicity in design is full of hidden merits. I recommend restricting GCM to 96-bit nonces.

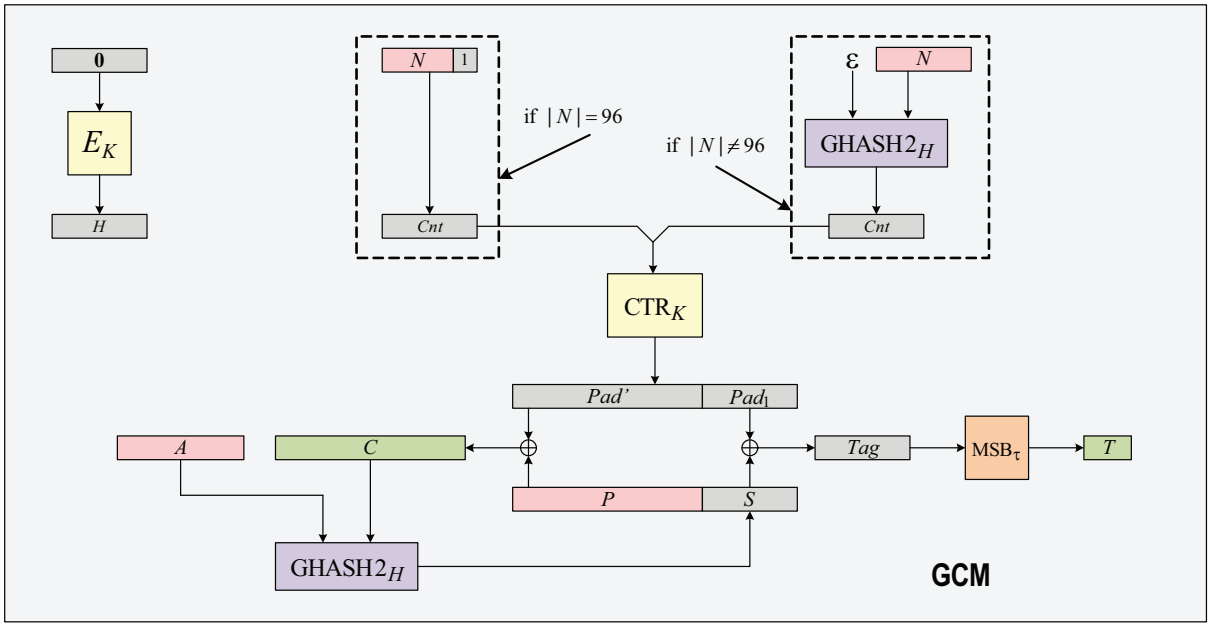
It is not just that permitting $|N| \neq 96$ runs contrary to simplicity of design and proof; it is that it does so with very little benefit. Allowing an arbitrary-length nonce in an AEAD scheme is nice, but it is not a sufficiently valuable benefit that one should be willing to pay for it with substantial added complexity.

I note that McGrew's own RFC 5116 [131] asserts that nonces *should* by 12 bytes. I do not think it unreasonable to more forcefully require this choice in the context of GCM, where nonces of other lengths are already treated as an unpleasant special case. And, in the end, everyone sticking to 12-byte nonces may help promote interoperability.

12.4.7 Even if the underlying AES mechanism is resistant to side-channel attacks, the use of tables to implement GCM's finite-field multiplication could potentially lead to cache-based timing attacks. See Käsper and Schwabe for work on trying to overcome this vulnerability by creating a constant-time GCM [108] (meaning fixed-time for strings of any particular length). While those authors are able to devise a reasonably fast (bitsliced) constant-time implementation of GCM, it runs at about 22.0 cycles/byte on their target x86-family platform—compared with (an impressive) 10.7 cycles/byte for a table-based GCM (using the same authors' bitsliced AES). The authors' work suggests that, in the absence of hardware-support for constant time $\mathrm{GF}(2^{128})$ multiplication (something like Intel's NIs), one will pay a substantial price, with GCM, for trying to resist timing attacks.

Concern for timing-attacks has resulted in OpenSSL choosing to support GCM (on non-Intel-NI platforms) using small tables only—256 bytes. This is a reasonable (if heuristic) measure to reduce the possibility of timing attacks [162].

In contrast to GCM, one would expect implementations of CCM to be constant-time (for messages of a given length) as long as the underlying blockcipher is; it is the use of large, key-dependent tables in a GCM implementation that makes it potentially



50 **algorithm** $\mathrm{GCM}_K^{N,A}(P)$ *GCM Encryption*
51 // $|P| \le 2^{39} - 256$, $|A| < 2^{64}$, $|N| > 0$, $|N| < 2^{64}$, $8 \mid |P|$, $8 \mid |A|$, $8 \mid |N|$
52 $H \leftarrow E_K(0^{128})$
53 **if** $|N| = 96$ **then** $Cnt \leftarrow N \parallel 0^{31}\,1$ **else** $Cnt \leftarrow \mathrm{GHASH2}_K(\varepsilon, N)$
54 $m \leftarrow \lceil |P|/128 \rceil$; **for** $i \leftarrow 0$ **to** m **do** $Y_i \leftarrow E_K(Cnt + i)$
55 $C \leftarrow P \oplus (Y_2 \parallel Y_3 \parallel Y_4 \parallel \cdots)$
56 $Tag \leftarrow Y_1 \oplus \mathrm{GHASH}_H(A, C)$
57 $T \leftarrow \mathrm{MSB}_\tau(Tag)$
58 **return** (C, T)

60 **algorithm** $\mathrm{GHASH2}_H(A, C)$ *Used internally*
61 $X \leftarrow A \parallel 0^i \parallel C \parallel 0^j \parallel |A|_{64} \parallel |C|_{64}$ for minimal $a, c \ge 0$ s.t. $128 \mid (|A| + i)$ and $128 \mid (|C| + j)$
62 $X_1 \cdots X_m \leftarrow X$ where $|X_i| = 128$
63 $Y \leftarrow 0^{128}$; **for** $i \leftarrow 1$ **to** m **do** $Y \leftarrow (Y \oplus X_i) \bullet H$
64 **return** Y

Figure 12.2: **Alternative decomposition of GCM.** We abstract away details of the hash function and revert to McGrew and Viega's original abstraction of it, renamed as GHASH2. Algorithm GCM.Decrypt (omitted) is unchanged. Value Y' in the illustration corresponds to $Y_2Y_3Y_4\cdots$ in the pseudocode. Routine $\mathrm{CTR}_K(Cnt)$ in the illustration returns $E_K(Cnt)E_K(Cnt+1)E_K(Cnt+2)\cdots$.

vulnerable.

All that said, we have not seen timing attacks demonstrated against any actual GCM implementation. In addition, if the underlying blockcipher resists timing attacks, then a timing attack on GCM might reasonably be expected to undermine authentication but not confidentiality.

12.4.8 Moving on to more expository matters, the original GCM documents [133, 137] defined a function GHASH that took in a key and a pair of strings and produced a 128-bit hash value. See Figure 12.2, where GCM is re-defined in terms of its "original" hash function, renamed as GHASH2 to distinguish it from NIST's version.

McGrew and Viega show that GHASH2 is xor-universal: for $(A, C) \neq (A', C')$ and $\Delta \in \{0,1\}^\tau$, the claim is that $\Pr_H[\mathrm{GHASH2}_H(A, C) \oplus \mathrm{GHASH2}_H(A', C') = \Delta] \le \epsilon$ with

$\epsilon = \lceil (m+1)/128 \rceil / 2^{128}$ and $m = \max\{|A|+|C|, |A'|+|C'|\}$. There are further important claims connected to GHASH2 and how it relates to the alternative construction of the initial counter value $Cnt = N \parallel 0^{31}1$ [133, Appendix A]. This non-interaction is at the heart of GCM's correctness, and one see it relatively easily with the GHASH2 abstraction.

In creating the SP 800-38D [64], NIST decided to redraw this abstraction boundary, regarding GHASH and not GHASH2 as the central abstraction boundary, the former a function of one string. The padding, concatenation, length-annotation are all done outside of this boundary. The result, as NIST recognizes [64, Footnote 1], is something that is no longer an xor-universal hash function.

I dislike this change. In drawing the abstraction boundary of GHASH2 the authors were not *only* trying to find a convenient boundary for modularizing GCM's description (which NIST's GHASH does fine); they are also laying into place the conceptual boundaries for understanding why the scheme work. At this level, the new abstraction works less well than the original one.

12.4.9 Getting to complaints more persnickety still, the name "GCM"—Galois/Counter Mode—has the word "mode" already contained within. As a consequence, one can't say "GCM mode" without implicitly saying "Galois/Counter-Mode mode." This has made for sometimes awkward discourse. Other unfortunate names include H for the subkey derived from K (an H looks like a hash-function—and there is one in GCM—not a string or key), J_0 for the initial counter value (can't explain that one at all), and *additional authenticated data* (AAD) (but sometimes *associated data*, the "standard" term and that of SP 800-38C [62]), sometimes *additional data*, and sometimes *additional, non-confidential data*—all for the exact same thing).

12.4.10 More serious, the exposition in the NIST spec seems to kind of "fall apart" in Sections 8 and 9, and in Appendix C. These sections stray from the goal of defining GCM, and make multiple incorrect or inscrutable statements. Here are some examples. *Page 18*: **The probability that the authenticated encryption function ever will be invoked with the same IV and the same key on two (or more) sets of input data shall be no greater than 2^{-32}** (here and later in this paragraph, imperatives are preserved in their original bold font). The probabilistic demand excludes use of almost all cryptographic PRGs (including those standardized by NIST), where no such guarantee is known. And I have no idea what the second part of the requirement (**sets of input data**) might mean. *Page 19*: **Any GCM key that is established among its intended users shall, with high probability, be fresh**. Freshness, while not having a single formal-model or computational-model definition, is rarely given a probabilistic interpretation; a key is not regarded as un-fresh, on a particular run, because of a rare probabilistic event. *Page 20*: It is impossible to determine if the (oddly named) *invocation field* can be empty. Additionally, notions like *devices* or *contexts* are fundamentally without foundation in this document. *Page 21*: **The total number of invocations of the authenticated encryption function shall not exceed 2^{32}, including all IV lengths and all instance of the authenticated encryption function with the given key**. This is incoherent. The request is to sum the number of encryption (and decryption?) calls and the IV lengths? David McGrew indicates [personal communications, 2/2011] that the spec should have said that "The total number of invocations of the authenticated encryption function shall not exceed 2^{32}, including [invocations with] all IV lengths and all instance[s] of the authenticated encryption function with the given key." The meaning of *implementation* a couple of

sentences later is also mysterious. *Page 22*: "In order to inhibit an unauthorized party from controlling or influencing the generation of IVs, ...". This statement seems to overlook that the nonce-based notion of AEAD is specifically *designed* to model the adversary having the ability to influence the IVs (subject to no-reuse). In real-world settings, there is often nothing one can do to stop the adversary from influencing the generation of IVs, which is what the formal model aims to address. *Page 23*: Discussions like what Section 9.2 has degenerated into belong in information appendices, not within the body of the specification document. *Pages 28–29*: An uneasy mix of formal requirements (the two tables) and informal recommendations (the rest). The policy suggested as item 1 on p. 28 (silently discard packets that decrypt to INVALID) is a reasonable policy in some settings, but tearing down the connection is also, sometimes, the right thing to do. Item 2 is not something the authors should be passing recommendations on: for a well-designed scheme, regardless of tag length, the user should feel free to place whatever he wants into the AD.

12.4.11 While the GCM mechanism is not itself covered by any known IP, there is at least one US patent [112] and three pending US patents [51, 76, 78] for GCM implementation techniques and basic usage of the mode. While the claims of these provisional and utility patents look, at a glance, rather suspect relative to the existing art to non-obviousness requirements, this is not always relevant in the disposition of patents and patent applications. The emergence of IP associated to natural implementations of GCM diminish one of its key advantages over OCB. We have not investigated the presence of IP outside the USA.

Looking now at some other authors' critiques, besides the tag-truncation attack already described, Ferguson raises a number of engineering-style concerns [67]. I have folded into the critique above those criticisms found to be of greatest merit.

Joux too writes critically of GCM, providing a key-recovery attack that works (it recovers H) if—contrary to requirements emphasized by SP 800-38D—an IV gets reused [104]. The attack probably motivated some of the strong language focused on avoiding IV-reuse. Joux points out that the consequences here of IV-reuse are *worse* than with CTR mode: IV reuse doesn't only compromise a particular ciphertext, it silently destroys all future authenticity. I concur, but do not find this surprising or overly problematic; Joux's attack is more at the level of a cautionary tale for what happens when one does not do as the instructions say to do. Joux also explains why an apparent optimization-attempt made by NIST in a draft version of the GCM spec was wrong. The optimization was "un-made" in NIST's final spec.

12.5. Provable security of GCM. The GCM authors make the following provable-security claims. For simplicity, we will assume the underlying blockcipher realizes a family of random permutations (passing to the complexity-theoretic setting is standard). Let's begin with privacy. McGrew and Viega assume a model in which an adversary $\mathcal{A}$ has either a GCM encryption oracle and a GCM decryption oracle, or else a random-bits oracle and a GCM decryption oracle. Nonces may not be repeated in encryption-oracle queries. The authors look at the difference in probabilities with which the adversary outputs "1" in each setting. The authors attribute this notion to Rogaway [178], but that work did not include the decryption oracle for defining privacy, effectively relying on equivalence results to banish it. The inclusion of decryption queries means that the privacy ends up inheriting the unpleasant tag-length dependency that one might expect to be associated only to the authenticity failure. This is purely artifactual.

Let $\mathcal{A}$ be an adversary that asks its oracles a sequence of queries where

- ℓ_N bounds the blocklength of every nonce N that is queried,
- ℓ bounds the sum of the block lengths for each (A, C) pair that arises during the adversaries queries,
- q bounds the total number of queries (either encryption or decryption queries), and
- σ bounds the total blocklength of all plaintext blocks processed (either queried or returned).

Then McGrew and Viega show that [133, Theorem 1]

$$\mathbf{Adv}^{\text{priv}}_{\text{GCM}[\text{Perm}(n),\tau]}(\mathcal{A}) \;\leq\; \frac{0.5(\sigma+2q)^2 + 0.5q(\sigma+2q)(\ell_N+1)}{2^n} + \frac{q(\ell+1)}{2^\tau}. \tag{12.1}$$

Here we write GCM[Perm$(n), \tau$] for the GCM scheme as it would be instantiated with an ideal n-bit blockcipher (E_K is a uniformly random permutation) and tag-length τ.

There are a couple of problems with this bound. One is that the $q^2\ell_N$ addend in the numerator is contributing $\Omega(\sigma^3/2^n)$ worst-case behavior. One instead wants an overall privacy bound of $O(\sigma^2/2^n)$ (ignoring the τ-dependency, which stems from the inclusion of decryption queries). Almost certainly this is an artifact of the analysis and the decision to express the bounds with respect to the particular parameters named above. The second problem is that the bound is bad if one is going to allow small τ. In particular, for τ having a value like $\tau = 32$ or even $\tau = 64$, it takes only a modest-sized q and ℓ to make $q\ell > 2^\tau$, whence there is no provable security remaining. We have already seen that the problem is not just a deficiency in the bound; the Ferguson attack shows this [67]. This is the reason underlying the restrictions on ℓ and q in [64, Appendix C].

As for authenticity, the authors' definition coincides with our own (§III.2). Using notation as before, the claim [133, Theorem 2] is that

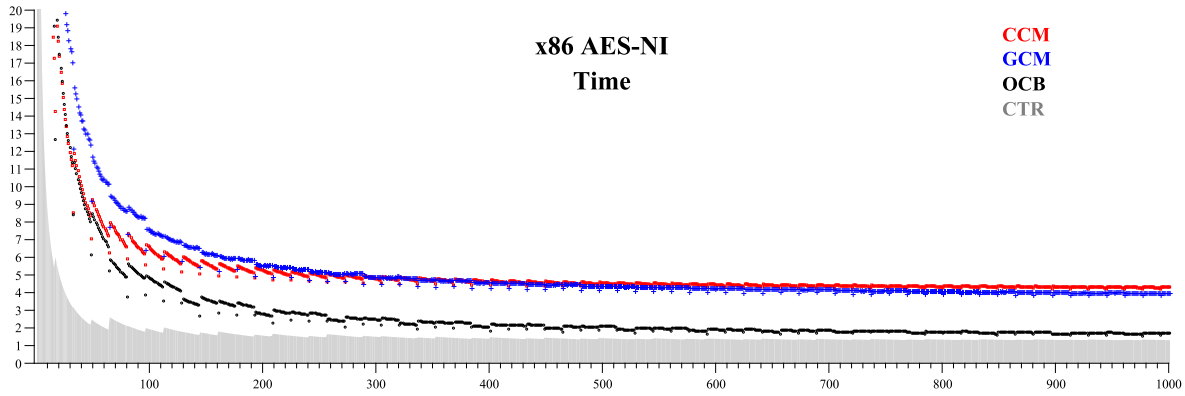
$$\mathbf{Adv}^{\text{auth}}_{\text{GCM}[\text{Perm}(n),\tau]}(\mathcal{A}) \;\leq\; \frac{0.5(\sigma+2q)^2 + 0.5q(\sigma+2q+1)(\ell_N+1)}{2^n} + \frac{q(\ell+1)}{2^\tau}. \tag{12.2}$$

The same comment applies: forgeries can—in principle and in practice—come quickly with short tags, when $q\ell \approx 2^\tau$. One also sees the same cubic dependency on σ: the first addend above is, at worst, about $\sigma^3/2^n$.

In short, there are provable security bounds for GCM, but they do not really say what one would wish them to say. And this is not entirely a weakness in the demonstrated bounds; there are, in part, corresponding attacks. Comparing the CCM bounds with the GCM bounds (see equations (11.13) and (11.14) from Chapter 11), the CCM bounds are much better. While the privacy bounds do not make a fair compassion, because of differing definitions used, the authenticity bounds end up different because of genuine differences in how the schemes do when tags are *not* the full 128-bits.

12.6. Truncating the tag. We have already treated this quite extensively in 12.3.1 and §12.5 but we wish to make a few additional comments. First, truncating message-authentication codes is a common thing to do. Arbitrary truncation is allowed for all of the ISO 9797-1 CBCMAC-derived mechanisms (Chapter 7) and for CMAC (Chapter 8). Tag truncation is likewise allowed in CCM (Chapter 11) and GCM. Unfortunately, the extent to which it is "safe" to truncate a MAC or AE-tag is not a simple story.

Second, when we speak of whether or not it is safe to truncate a tag, we are *not* simply speaking of the obvious matter that one can forge a τ-bit tag with probability $2^{-\tau}$ by just



x86 i5-650 AES-NI				
Mode	T_{4K}	T_{IPI}	Size	Init
CCM	4.17	4.57	512	265
GCM	3.73	4.53	656	337
OCB	1.48	2.08	544	251
CTR	1.27	1.37	244	115

x86 i5-650 Käsper-Schwabe				
Mode	T_{4K}	T_{IPI}	Size	Init
GCM-bitsliced	22.4	26.7	1456	3780
GCM-8K	10.9	15.2	9648	2560
OCB	8.28	13.4	3008	3390
CTR	7.74	8.98	1424	1180

ARM Cortex-A8				
Mode	T_{4K}	T_{IPI}	Size	Init
CCM	51.3	53.7	512	1390
GCM-256	50.8	53.9	656	1180
OCB	29.3	31.5	672	1920
CTR	25.4	25.9	244	236

Figure 12.3: **Empirical performance of some AE modes.** The data is extracted from Krovetz and Rogaway [118]. The findings are very different what McGrew and Viega report [133, 134, 137]. **Top:** Speed (cpb) verses message length (1–1000 bytes) on an Intel x86 i5 processor, using code that makes use of Intel's NIs. **Bottom:** For each architecture we give speed (cpb) to encrypt 4096 byte messages, time to encrypt a weighted basket of message lengths (IPI, also in cpb), size of the implementation's context (in bytes), and time to initialize key-dependent values (in CPU cycles).

guessing a random tag. That "attack" is easily understood, and the security architect is able to ascertain whether, in a given context, a $2^{-\tau}$ probability of an adversary guessing a correct tag is a problem. What we are speaking of here is whether or not truncation of tags is guaranteed to have the "expected" effect, where this entails that the chance of succeeding in forging, in each and every forgery attempt, shouldn't stray much from $2^{-\tau}$ until the adversary has asked an unreasonable number of queries or material, or has computed for an unreasonable amount of time. This is not a formal definition. See Wang, Feng, Lin, and Wu [197] (following Krovetz and Rogaway [117]) and Black and Cochran [35] for a more systematic treatment.

Third, we would emphasize that, to date, there has been no published definition for AEAD that guarantees that arbitrarily truncating tags does what one wants. Thus when we say that encrypt-then-PRF generic composition does the job, or that a mode like CCM seems to be correct in this regard, the statements can only be informal, for now, lacking the needed justification.

NIST was clearly mindful of the short-tag issue; the tables of [64, Appendix C], especially for 32-bit lengths, mandate rather severe requirements for re-keying. I suggest, however, that disallowing 32- and 64-bit tags is a more reasonable approach for dealing with the problem.

12.7. Software performance. While the software performance of GCM can be quite good, it can also be quite bad. This statement may sound vacuous (*every* algorithms can be implemented poorly or well), but the intended meaning is not: I am suggesting that the software performance of GCM is *particularly* sensitive to nontrivial implementation choices. Among these choices: how large one wants to make internal tables; how much time one wants to spend on key-setup; the extent to which one aims for timing-attack resistance; and to what extent one is willing to assume that most messages are long.

One of the selling-points for GCM was McGrew and Viega's claim [133, 134, 137] that, in software, the mode is considerably faster than CCM, and about as fast OCB. In a careful multi-platform timing study done with Ted Krovetz [118], we arrive at quite different findings: that CCM and GCM performance are usually similar, while OCB is usually much faster than both. For summary performance findings from our paper, see Figure 12.3. We compare CCM, GCM, OCB and, as a baseline, CTR. The OCB version that we report on here is the original one [182]. A still faster variant now exists as well [118].

In the **leftmost** table at Figure 12.3, an x86 i5 platform is used and the code employs Intel's new AES and carryless multiply NIs. This obviates issues of table size. The CCM and GCM implementations are from a developmental version of OpenSSL, which has the fastest times known [162]. For the **middle** table we report how well one can do, on the same platform, without the NIs, using either a bitsliced (constant-time) or a table-based GCM, and a bitsliced AES, all of which were authored by Käsper and Schwabe [108]. We add in our own OCB code (which calls down to the Käsper-Schwabe AES). CCM is not included in this test, but would perform terribly, since the bitslided AES-code needs to compute eight AES ECB computations at a time, and, when CBC-chaining in CCM, we have only one AES we can do at a time. The **rightmost** table, giving ARM data, uses OpenSSL assembly code for CCM, CTR, and GCM, plus our own C code for OCB. The GCM code is designed to be timing-attack resistant, and is therefore somewhat slow. This does not make comparisons irrelevant; CCM (and OCB) are also timing-attack resistant, making the playing field level.

We used a Debian Linux 6.0 with kernel 2.6.35 and GCC 4.5.1. Compilation is done with `-O3` optimization, `-mcpu` or `-march` set according to the host processor, and `-m64` to force 64-bit compilation when needed. The IPI measure ("Internet Performance Index"), due to McGrew and Viega [137], is a weighted sum of performance figures for messages of some specified lengths. The intent is to have a measure that dissuades one from focusing just on peak performance, as speed on short messages can matter a lot in shaping real-world performance.

12.8. Concluding remarks. While much of this chapter has been critical of GCM, one should keep things in perspective: the mode is popular, efficient, and enjoys good provable-security results if tags are long. The most serious problem—the use of short tags—can be outlawed. Another important issue—the danger that the user will, contrary to instructions, reuse a nonce—is certainly a valid concern, but it is endemic to many modes of operation, and it is costly to deal with well, as two passes will then be needed over the data [183]. Ultimately, we find Ferguson's advise—"Do not use GCM" [67, p. 9]—as rather too strident given what he shows and what is currently know. Instead, we would say that the mode—like most—needs to be used with some care.

Bibliography

[1] Abdalla, M., and Bellare, M. Increasing the lifetime of a key: a comparative analysis of the security of re-keying techniques. In *Advances in Cryptology – ASIACRYPT 2000* (Kyoto, Japan, Dec. 3–7, 2000), T. Okamoto, Ed., vol. 1976 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 546–559.

[2] Alkassar, A., Geraldy, A., Pfitzmann, B., and Sadeghi, A.-R. Optimized self-synchronizing mode of operation. In *Fast Software Encryption – FSE 2001* (Yokohama, Japan, Apr. 2–4, 2001), M. Matsui, Ed., vol. 2355 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 78–91.

[3] American National Standards Institute. ANSI X9.19. Financial institution retail message authentication. American National Standards Institute, Aug. 1986.

[4] American National Standards Institute. ANSI X9.9-1986 (revised), Financial institution message authentication (wholesale). American Nationalal Standards Institute, Apr. 1986.

[5] American National Standards Institute. ANSI X9.52. Triple Data Encryption Algorithm Modes of Operation. American National Standards Institute, Jan. 1998.

[6] An, J. H., and Bellare, M. Constructing VIL-MACs from FIL-MACs: Message authentication under weakened assumptions. In *Advances in Cryptology – CRYPTO'99* (Santa Barbara, CA, USA, Aug. 15–19, 1999), M. J. Wiener, Ed., vol. 1666 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 252–269.

[7] Aoki, K., and Lipmaa, H. Fast implementations of AES candidaates. In *AES Candidate Conference 2000* (2000), pp. 106–120.

[8] Ball, M. Cryptographic protection of data on block-oriented storage devices. CRYPTO 2008 rump-session talk, slides available from http://siswg.net, 2008.

[9] Barak, B., Haitner, I., Hofheinz, D., and Ishai, Y. Bounded key-dependent message security. In *Advances in Cryptology – EUROCRYPT 2010* (French Riviera, May 30 – June 3, 2010), H. Gilbert, Ed., vol. 6110 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 423–444.

[10] Barker, W. NIST Special Publication 800-67, version 1.1. Recommendation for the Triple Data Encryption Algorithm (TDEA) block cipher, May 2008.

[11] Bellare, M. New proofs for NMAC and HMAC: Security without collision-resistance. In *Advances in Cryptology – CRYPTO 2006* (Santa Barbara, CA, USA, Aug. 20–24, 2006), C. Dwork, Ed., vol. 4117 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 602–619.

[12] Bellare, M., Boldyreva, A., Knudsen, L., and Namprempre, C. Online ciphers and the hash-CBC construction. In *Advances in Cryptology – CRYPTO 2001* (Santa Barbara, CA, USA, Aug. 19–23, 2001), J. Kilian, Ed., vol. 2139 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 292–309.

[13] Bellare, M., Canetti, R., and Krawczyk, H. Keying hash functions for message authentication. In *Advances in Cryptology – CRYPTO'96* (Santa Barbara, CA, USA, Aug. 18–22, 1996), N. Koblitz, Ed., vol. 1109 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 1–15.

[14] Bellare, M., Desai, A., Jokipii, E., and Rogaway, P. A concrete security treatment of symmetric encryption. In *38th Annual Symposium on Foundations of Computer Science* (Miami Beach, Florida, Oct. 19–22, 1997), IEEE Computer Society Press, pp. 394–403.

[15] Bellare, M., Goldreich, O., and Goldwasser, S. Incremental cryptography and application to virus protection. In *27th Annual ACM Symposium on Theory of Computing* (Las Vegas, Nevada, USA, May 29 – June 1, 1995), ACM Press, pp. 45–56.

[16] Bellare, M., Goldreich, O., and Mityagin, A. The power of verification queries in message authentication and authenticated encryption. Cryptology ePrint Archive, Report 2004/309, Nov. 2004.

[17] Bellare, M., Guérin, R., and Rogaway, P. XOR MACs: New methods for message authentication using finite pseudorandom functions. In *Advances in Cryptology – CRYPTO'95* (Santa Barbara, CA, USA, Aug. 27–31, 1995), D. Coppersmith, Ed., vol. 963 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 15–28.

[18] Bellare, M., Kilian, J., and Rogaway, P. The security of cipher block chaining. In *Advances in Cryptology – CRYPTO'94* (Santa Barbara, CA, USA, Aug. 21–25, 1994), Y. Desmedt, Ed., vol. 839 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 341–358.

[19] Bellare, M., Kilian, J., and Rogaway, P. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences 61*, 3 (2000), 362–399.

[20] Bellare, M., and Kohno, T. Hash function balance and its impact on birthday attacks. In *Advances in Cryptology – EUROCRYPT 2004* (Interlaken, Switzerland, May 2–6, 2004), C. Cachin and J. Camenisch, Eds., vol. 3027 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 401–418.

[21] Bellare, M., Krovetz, T., and Rogaway, P. Luby-Rackoff backwards: Increasing security by making block ciphers non-invertible. In *Advances in Cryptology – EUROCRYPT'98* (Espoo, Finland, May 31 – June 4, 1998), K. Nyberg, Ed., vol. 1403 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 266–280.

[22] Bellare, M., and Namprempre, C. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *Advances in Cryptology – ASIACRYPT 2000* (Kyoto, Japan, Dec. 3–7, 2000), T. Okamoto, Ed., vol. 1976 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 531–545.

[23] Bellare, M., and Namprempre, C. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *Journal of Cryptology 21*, 4 (Oct. 2008), 469–491.

[24] Bellare, M., Pietrzak, K., and Rogaway, P. Improved security analyses for CBC MACs. In *Advances in Cryptology – CRYPTO 2005* (Santa Barbara, CA, USA, Aug. 14–18, 2005), V. Shoup, Ed., vol. 3621 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 527–545.

[25] Bellare, M., and Rogaway, P. Entity authentication and key distribution. In *Advances in Cryptology – CRYPTO'93* (Santa Barbara, CA, USA, Aug. 22–26, 1994), D. R. Stinson, Ed., vol. 773 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 232–249.

[26] Bellare, M., and Rogaway, P. On the construction of variable-input-length ciphers. In *Fast Software Encryption – FSE'99* (Rome, Italy, Mar. 24–26, 1999), L. Knudsen, Ed., vol. 1636 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 231–244.

[27] Bellare, M., and Rogaway, P. The security of triple encryption and a framework for code-based game-playing proofs. In *Advances in Cryptology – EUROCRYPT 2006* (St. Petersburg, Russia, May 28 – June 1, 2006), S. Vaudenay, Ed., vol. 4004 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 409–426.

[28] Bellovin, S. Problem areas for the ip security protocols. In *Proceedings of the Sixth USENIX Security Symposium* (1996), USENIX.

[29] Bernstein, D. A short proof of unpredictability of cipher block chaining, 2005. Unpublished manuscript available from the author's webpage.

[30] Bernstein, D. J., and Schwabe, P. New AES software speed records. In *Progress in Cryptology - INDOCRYPT 2008: 9th International Conference in Cryptology in India* (Kharagpur, India, Dec. 14–17, 2008), D. R. Chowdhury, V. Rijmen, and A. Das, Eds., vol. 5365 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 322–336.

[31] Bharadwaj, V., and Ferguson, N. Public comments on the XTS-AES mode. Collected email comments released by NIST, available from their web page, 2008.

[32] Biham, E., and Shamir, A. Differential cryptanalysis of Snefru, Khafre, REDOC-II, LOKI and Lucifer. In *Advances in Cryptology – CRYPTO'91* (Santa Barbara, CA, USA, Aug. 11–15, 1992), J. Feigenbaum, Ed., vol. 576 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 156–171.

[33] Biryukov, A., and Khovratovich, D. Related-key cryptanalysis of the full AES-192 and AES-256. In *Advances in Cryptology – ASIACRYPT 2009* (Tokyo, Japan, Dec. 6–10, 2009), M. Matsui, Ed., vol. 5912 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 1–18.

[34] Biryukov, A., Khovratovich, D., and Nikolic, I. Distinguisher and related-key attack on the full AES-256. In *Advances in Cryptology – CRYPTO 2009* (Santa Barbara, CA, USA, Aug. 16–20, 2009), S. Halevi, Ed., vol. 5677 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 231–249.

[35] Black, J., and Cochran, M. MAC reforgeability. In *Fast Software Encryption – FSE 2009* (Leuven, Belgium, Feb. 22–25, 2009), O. Dunkelman, Ed., vol. 5665 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 345–362.

[36] Black, J., Cochran, M., and Highland, T. A study of the MD5 attacks: Insights and improvements. In *Fast Software Encryption – FSE 2006* (Graz, Austria, Mar. 15–17, 2006), M. Robshaw, Ed., vol. 4047 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 262–277.

[37] Black, J., Halevi, S., Krawczyk, H., Krovetz, T., and Rogaway, P. UMAC: Fast and secure message authentication. In *Advances in Cryptology – CRYPTO'99* (Santa Barbara, CA, USA, Aug. 15–19, 1999), M. J. Wiener, Ed., vol. 1666 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 216–233.

[38] Black, J., and Rogaway, P. CBC MACs for arbitrary-length messages: The three-key constructions. In *Advances in Cryptology – CRYPTO 2000* (Santa Barbara, CA, USA, Aug. 20–24, 2000), M. Bellare, Ed., vol. 1880 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 197–215.

[39] Black, J., and Rogaway, P. CBC MACs for arbitrary-length messages: The three-key constructions. *Journal of Cryptology 18*, 2 (Apr. 2005), 111–131.

[40] Black, J., Rogaway, P., and Shrimpton, T. Encryption-scheme security in the presence of key-dependent messages. In *SAC 2002: 9th Annual International Workshop on Selected Areas in Cryptography* (St. John's, Newfoundland, Canada, Aug. 15–16, 2003), K. Nyberg and H. M. Heys, Eds., vol. 2595 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 62–75.

[41] Black, J., Rogaway, P., Shrimpton, T., and Stam, M. An analysis of the blockcipher-based hash functions from PGV. *Journal of Cryptology 23*, 4 (Oct. 2010), 519–545.

[42] Black, J., and Urtubia, H. Side-channel attacks on symmetric encryption schemes: The case for authenticated encryption. In *USENIX Security Symposium* (2002), D. Boneh, Ed., USENIX, pp. 327–338.

[43] Bleichenbacher, D. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In *Advances in Cryptology – CRYPTO'98* (Santa Barbara, CA, USA, Aug. 23–27, 1998), H. Krawczyk, Ed., vol. 1462 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 1–12.

[44] Boneh, D., Halevi, S., Hamburg, M., and Ostrovsky, R. Circular-secure encryption from decision diffie-hellman. In *Advances in Cryptology – CRYPTO 2008* (Santa Barbara, CA, USA, Aug. 17–21, 2008), D. Wagner, Ed., vol. 5157 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 108–125.

[45] Bosselaers, A., and Preneel, B., Eds. *Integrity Primitives for Secure Information Systems, Final Report of RACE Integrity Primitives Evaluation RIPE-RACE 1040*, vol. 1007 of *Lecture Notes in Computer Science*. Springer, 1995.

[46] Brassard, G. On computationally secure authentication tags requiring short secret shared keys. In *Advances in Cryptology – CRYPTO'82* (Santa Barbara, CA, USA, 1983), D. Chaum, R. L. Rivest, and A. T. Sherman, Eds., Plenum Press, New York, USA, pp. 79–86.

[47] Brincat, K., and Mitchell, C. New CBC-MAC forgery attacks. In *ACISP 05: 10th Australasian Conference on Information Security and Privacy* (Brisbane, Queensland, Australia, 2001), V. Varadharajan and Y. Mu, Eds., vol. 2119 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 3–14.

[48] Camenisch, J., and Lysyanskaya, A. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Advances in Cryptology – EUROCRYPT 2001* (Innsbruck, Austria, May 6–10, 2001), B. Pfitzmann, Ed., vol. 2045 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 93–118.

[49] Canvel, B., Hiltgen, A. P., Vaudenay, S., and Vuagnoux, M. Password interception in a SSL/TLS channel. In *Advances in Cryptology – CRYPTO 2003* (Santa Barbara, CA, USA, Aug. 17–21, 2003), D. Boneh, Ed., vol. 2729 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 583–599.

[50] Carter, L., and Wegman, M. Universal hash functions. *J. of Computer and System Sciences*, 18 (1979), 143–154.

[51] Chen, L., and Buckingham, J. Authenticated encryption method and apparatus, Apr. 2008. US Patent Application 20060126835.

[52] Clunie, D. Public comments on the XTS-AES mode. Collected email comments released by NIST, available from their web page, 2008.

[53] Contini, S., and Yin, Y. Forgery and partial key-recovery attacks on HMAC and NMAC using hash collisions. In *Advances in Cryptology – ASIACRYPT 2006* (Shanghai, China, Dec. 3–7, 2006), X. Lai and K. Chen, Eds., vol. 4284 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 37–53.

[54] Coppersmith, D., Knudsen, L., and Mitchell, C. Key recovery and forgery attacks on the MacDES MAC algorithm. In *Advances in Cryptology – CRYPTO 2000* (Santa Barbara, CA, USA, Aug. 20–24, 2000), M. Bellare, Ed., vol. 1880 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 184–196.

[55] Coppersmith, D., and Mitchell, C. Attacks on MacDES MAC algorithm. *Electronics Letters 35*, 19 (1999), 1626–1627.

[56] Degabriele, J. P., and Paterson, K. G. Attacking the IPsec standards in encryption-only configurations. In *2007 IEEE Symposium on Security and Privacy* (Oakland, California, USA, May 20–23, 2007), IEEE Computer Society Press, pp. 335–349.

[57] den Boer, B., and Bosselaers, A. Collisions for the compressin function of MD5. In *Advances in Cryptology – EUROCRYPT'93* (Lofthus, Norway, May 23–27, 1993), T. Helleseth, Ed., vol. 765 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 293–304.

[58] Diffie, W., and Hellman, M. Privacy and authentication: an introduction to cryptography. *Proceedings of the IEEE 67* (1979), 397–427.

[59] Dodis, Y., Gennaro, R., Håstad, J., Krawczyk, H., and Rabin, T. Randomness extraction and key derivation using the CBC, cascade and HMAC modes. In *Advances in Cryptology – CRYPTO 2004* (Santa Barbara, CA, USA, Aug. 15–19, 2004), M. Franklin, Ed., vol. 3152 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 494–510.

[60] Dolev, D., Dwork, C., and Naor, M. Nonmalleable cryptography. *SIAM Journal on Computing 30*, 2 (2000), 391–437.

[61] Dworkin, M. NIST Special Publication 800-38A. Recommendation for block cipher modes of operation: Modes and techniques, Dec. 2001.

[62] Dworkin, M. NIST Special Publication 800-38C. Recommendation for block cipher modes of operation: The CCM mode for authentication and confidentiality, May 2004.

[63] Dworkin, M. NIST Special Publication 800-38B. Recommendation for block cipher modes of operation: The CMAC mode for authentication, May 2005.

[64] Dworkin, M. NIST Special Publication 800-38D. Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM and GMAC), Nov. 2007.

[65] Dworkin, M. NIST Special Publication 800-38E: Recommendation for block cipher modes of operation: The XTS-AES mode of confidentiality on storage devices, Jan. 2010.

[66] Even, S., and Mansour, Y. A construction of a cipher from a single pseudorandom permutation. In *Advances in Cryptology – ASIACRYPT'91* (Fujiyoshida, Japan, Nov. 11–14, 1991), H. Imai, R. L. Rivest, and T. Matsumoto, Eds., vol. 739 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 210–224.

[67] Ferguson, N. Authentication weaknesses in GCM. Manuscript, available from NIST's webpage, 2005.

[68] Ferguson, N. Personal communication, Feb. 2011.

[69] Fischlin, M. Security of NMAC and HMAC based on non-malleability. In *Topics in Cryptology – CT-RSA 2008* (San Francisco, CA, USA, Apr. 7–11, 2008), T. Malkin, Ed., vol. 4964 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 138–154.

[70] Fouque, P.-A., Leurent, G., and Nguyen, P. Q. Full key-recovery attacks on HMAC/NMAC-MD4 and NMAC-MD5. In *Advances in Cryptology – CRYPTO 2007* (Santa Barbara, CA, USA, Aug. 19–23, 2007), A. Menezes, Ed., vol. 4622 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 13–30.

[71] Fouque, P.-A., Martinet, G., Valette, F., and Zimmer, S. On the security of the CCM encryption mode and of a slight variant. In *ACNS 08: 6th International Conference on Applied Cryptography and Network Security* (New York, NY, USA, June 3–6, 2008), S. M. Bellovin, R. Gennaro, A. D. Keromytis, and M. Yung, Eds., vol. 5037 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 411–428.

[72] Fouque, P.-A., Pointcheval, D., and Zimmer, S. HMAC is a randomness extractor and applications to TLS. In *ASIACCS 08: 3rd Conference on Computer and Communications Security* (Tokyo, Japan, Mar. 18–20, 2008), M. Abe and V. Gligor, Eds., ACM Press, pp. 21–32.

[73] Gligor, V., and Donescu, P. Fast encryption and authentication: XCBC encryption and XECB authentication modes. In *Fast Software Encryption – FSE 2001* (Yokohama, Japan, Apr. 2–4, 2001), M. Matsui, Ed., vol. 2355 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 92–108.

[74] Goldwasser, S., and Micali, S. Probabilistic encryption. *Journal of Computer and System Sciences 28*, 2 (1984), 270–299.

[75] Goldwasser, S., Micali, S., and Rackoff, C. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing 18*, 1 (1989), 186–208.

[76] Gueron, S. Speeding up Galois Counter Mode (GCM) computations, June 2007. US Patent Application 20080240423.

[77] Gueron, S. Intel Advanced Encryption Standard (AES) instructions set — rev 3. White paper, avialable from www.intel.com, Jan. 2010.

[78] Gueron, S., and Kounavis, M. Using a single instruction multiple data (SIMD) instruction to speed up Galois Counter Mode (GCM) computation, Dec. 2009. US Patent Application 20090310775.

[79] Gueron, S., and Kounavis, M. Intel carry-less multiplication instruction and its usage for computing the GCM mode (revision 2). White paper, avialable from www.intel.com. List visited Jan 2011., May 2010.

[80] Halevi, S. EME*: Extending EME to handle arbitrary-length messages with associated data. In *Progress in Cryptology - INDOCRYPT 2004: 5th International Conference in Cryptology in India* (Chennai, India, Dec. 20–22, 2004), A. Canteaut and K. Viswanathan, Eds., vol. 3348 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 315–327.

[81] Halevi, S. Storage encryption: A cryptographer's view. Invited talk at the *Sixth Conference on Security and Cryptography for Networks* (SCN 2008). Slides available from URL http://people.csail.mit.edu/shaih/, 2008.

[82] Halevi, S., and Rogaway, P. A tweakable enciphering mode. In *Advances in Cryptology – CRYPTO 2003* (Santa Barbara, CA, USA, Aug. 17–21, 2003), D. Boneh, Ed., vol. 2729 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 482–499.

[83] Halevi, S., and Rogaway, P. A parallelizable enciphering mode. In *Topics in Cryptology – CT-RSA 2004* (San Francisco, CA, USA, Feb. 23–27, 2004), T. Okamoto, Ed., vol. 2964 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 292–304.

[84] Hall, C., Wagner, D., Kelsey, J., and Schneier, B. Building PRFs from PRPs. In *Advances in Cryptology – CRYPTO'98* (Santa Barbara, CA, USA, Aug. 23–27, 1998), H. Krawczyk, Ed., vol. 1462 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 370–389.

[85] Handschuh, H., and Preneel, B. Key-recovery attacks on universal hash function based MAC algorithms. In *Advances in Cryptology – CRYPTO 2008* (Santa Barbara, CA, USA, Aug. 17–21, 2008), D. Wagner, Ed., vol. 5157 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 144–161.

[86] IEEE 1619 SISWG. Security in Storage Working Group, P1619 Narrow-Block. Web page maintained by the IEEE, URL: http://siswg.net/, last visited January 2011.

[87] Institute of Electrical and Electronics Engineers. IEEE Std. 802.11-1999, IEEE standard for telecommunications and informnation exchange between systems — LAN/MAN specific requirements — Part II: Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications. IEEE Press, 1999.

[88] Institute of Electrical and Electronics Engineers. IEEE Std.— 802.1AE-2006, IEEE Standard for local and metropolitan area networks – Media Access Control (MAC) security. IEEE Press, 2006.

[89] Institute of Electrical and Electronics Engineers. IEEE Std. 1619.1-2007, IEEE standard for authenticated encryption with length expansion for storage devices. IEEE Press, 2007.

[90] Institute of Electrical and Electronics Engineers. IEEE Std. 1619-2007, IEEE standard for cryptographic protection of data on block-oriented storage devices. IEEE Press, Apr. 2008.

[91] International Organization for Standardization and International Electrotechnical Commission. ISO/IEC 9797-1:1999, Information technology — Security techniques — Message Authentication Codes (MACs) — Part 1: Mechanisms using a block cipher. International Standard, 1999.

[92] International Organization for Standardization and International Electrotechnical Commission. ISO/IEC 10116:2006, Information technology — Security techniques — Modes of Operation for an n-bit block cipher. International Standard, 2006.

[93] International Organization for Standardization and International Electrotechnical Commission. ISO/IEC FDIS 9797-1:2010(E), Information technology — Security techniques — Message Authentication Codes (MACs) — Part 1: Mechanisms using a block cipher. Final Draft International Standard, 2010.

[94] Iwata, T. Comments on "On the security of XCBC, TMAC and OMAC" by Mitchell, Sept. 2003. Manuscript, availabe from NIST's website.

[95] Iwata, T. New blockcipher modes of operation with beyond the birthday bound security. In *Fast Software Encryption – FSE 2006* (Graz, Austria, Mar. 15–17, 2006), M. Robshaw, Ed., vol. 4047 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 310–327.

[96] Iwata, T., and Kurosawa, K. OMAC: One-key CBC MAC. In *Fast Software Encryption – FSE 2003* (Lund, Sweden, Feb. 24–26, 2003), T. Johansson, Ed., vol. 2887 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 129–153.

[97] Iwata, T., and Kurosawa, K. Stronger security bounds for OMAC, TMAC, and XCBC. In *Progress in Cryptology - INDOCRYPT 2003: 4th International Conference in Cryptology in India* (New Delhi, India, Dec. 8–10, 2003), T. Johansson and S. Maitra, Eds., vol. 2904 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 402–415.

[98] Iwata, T., and Kurosawa, K. OMAC: One-key CBC MAC — Addendum. Manuscript, available from NIST's website, 2005.

[99] J. Song, R. Poovendran, and J. Lee. The AES-CMAC-96 algorithm and its use with IPsec. RFC 4494, June 2006.

[100] J. Song, R. Poovendran, J. Lee, and T. Iwata. The AES-CMAC algorithm. RFC 4493, June 2006.

[101] Jaulmes, É., Joux, A., and Valette, F. On the security of randomized CBC-MAC beyond the birthday paradox limit: A new construction. In *Fast Software Encryption – FSE 2002* (Leuven, Belgium, Feb. 4–6, 2002), J. Daemen and V. Rijmen, Eds., vol. 2365 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 237–251.

[102] Jia, K., Wang, X., Yuan, Z., and Xu, G. Distinguishing and second-preimage attacks on CBC-like MACs. In *CANS 09: 8th International Conference on Cryptology and Network Security* (Kanazawa, Japan, Dec. 12–14, 2009), J. A. Garay, A. Miyaji, and A. Otsuka, Eds., vol. 5888 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 349–361.

[103] Jonsson, J. On the security of CTR + CBC-MAC. In *SAC 2002: 9th Annual International Workshop on Selected Areas in Cryptography* (St. John's, Newfoundland, Canada, Aug. 15–16, 2003), K. Nyberg and H. M. Heys, Eds., vol. 2595 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 76–93.

[104] Joux, A. Authentication failures in [sic] NIST version of GCM, 2006. Manuscript, available from NIST's webpage.

[105] Joux, A., Poupard, G., and Stern, J. New attacks against standardized MACs. In *Fast Software Encryption – FSE 2003* (Lund, Sweden, Feb. 24–26, 2003), T. Johansson, Ed., vol. 2887 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 170–181.

[106] Jutla, C. Encryption modes with almost free message integrity. In *Advances in Cryptology – EUROCRYPT 2001* (Innsbruck, Austria, May 6–10, 2001), B. Pfitzmann, Ed., vol. 2045 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 529–544.

[107] Jutla, C. Encryption modes with almost free message integrity. *Journal of Cryptology 21*, 4 (Oct. 2008), 547–578.

[108] Käsper, E., and Schwabe, P. Faster and timing-attack resistant AES-GCM. In *Cryptographic Hardware and Embedded Systems – CHES 2009* (Lausanne, Switzerland, Sept. 6–9, 2009), C. Clavier and K. Gaj, Eds., vol. 5747 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 1–17.

[109] Katz, J., and Yung, M. Unforgeable encryption and chosen ciphertext secure modes of operation. In *Fast Software Encryption – FSE 2000* (New York, NY, USA, Apr. 10–12, 2000), B. Schneier, Ed., vol. 1978 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 284–299.

[110] Kilian, J., and Rogaway, P. How to protect DES against exhaustive key search (an analysis of DESX). *Journal of Cryptology 14*, 1 (2001), 17–35.

[111] Kim, J., Biryukov, A., Preneel, B., and Hong, S. On the security of HMAC and NMAC based on HAVAL, MD4, MD5, SHA-0 and SHA-1 (extended abstract). In *SCN 06: 5th International Conference on Security in Communication Networks* (Maiori, Italy, Sept. 6–8, 2006), R. D. Prisco and M. Yung, Eds., vol. 4116 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 242–256.

[112] Kim, K., Han, K., Yoo, T., and Kwon, Y. High-speed GCM-AES block cipher apparatus and method, Nov. 2010. US Patent 7,840,003.

[113] Knudsen, L. Chosen-text attack on CBC-MAC. *Electronics Letters 33*, 1 (1998), 48–49.

[114] Knudsen, L., and Preneel, B. MacDES: MAC algorithm based on DES. *Electronics Letters 34* (1998), 871–873.

[115] Kohno, T., Viega, J., and Whiting, D. CWC: A high-performance conventional authenticated encryption mode. In *Fast Software Encryption – FSE 2004* (New Delhi, India, Feb. 5–7, 2004), B. Roy and W. Meier, Eds., vol. 3017 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 408–426.

[116] Krawczyk, H. LFSR-based hashing and authentication. In *Advances in Cryptology – CRYPTO'94* (Santa Barbara, CA, USA, Aug. 21–25, 1994), Y. Desmedt, Ed., vol. 839 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 129–139.

[117] Krovetz, T., and Rogaway, P. Variationally universal hashing. *Information Processing Letters 100*, 1 (2006), 36–39.

[118] Krovetz, T., and Rogaway, P. The software performance of authenticated-encryption modes. To appear at FSE 2011, 2011.

[119] Kurosawa, K., and Iwata, T. TMAC: Two-key CBC MAC. In *Topics in Cryptology – CT-RSA 2003* (San Francisco, CA, USA, Apr. 13–17, 2003), M. Joye, Ed., vol. 2612 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 33–49.

[120] Lee, E., Chang, D., Kim, J., Sung, J., and Hong, S. Second preimage attack on 3-pass HAVAL and partial key-recovery attacks on HMAC/NMAC-3-pass HAVAL. In *Fast Software Encryption – FSE 2008* (Lausanne, Switzerland, Feb. 10–13, 2008), K. Nyberg, Ed., vol. 5086 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 189–206.

[121] Lipmaa, H. IDEA: A cipher for multimedia architectures? In *SAC 1998: 5th Annual International Workshop on Selected Areas in Cryptography* (Kingston, Ontario, Canada, Aug. 17–18, 1999), S. E. Tavares and H. Meijer, Eds., vol. 1556 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 248–263.

[122] Lipmaa, H., Rogaway, P., and Wagner, D. CTR-mode encryption. Comments sent to NIST and available their web page, or from Rogaway's, Sept. 2000.

[123] Liskov, M., and Minematsu, K. Comments on XTS-AES. Comments to NIST, available from their web page, Sept. 2008.

[124] Liskov, M., Rivest, R., and Wagner, D. Tweakable block ciphers. In *Advances in Cryptology – CRYPTO 2002* (Santa Barbara, CA, USA, Aug. 18–22, 2002), M. Yung, Ed., vol. 2442 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 31–46.

[125] Locktyukhin, M. Improving the performance of the Secure Hash Algorithm (SHA-1). White paper, avialable from www.intel.com, Mar. 2010.

[126] Luby, M., and Rackoff, C. How to construct pseudo-random permutations from pseudo-random functions (abstract). In *Advances in Cryptology – CRYPTO'85* (Santa Barbara, CA, USA, Aug. 18–22, 1986), H. C. Williams, Ed., vol. 218 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, p. 447.

[127] Luby, M., and Rackoff, C. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing 17*, 2 (1988).

[128] Luby, M., and Rackoff, C. A study of password security. In *Advances in Cryptology – CRYPTO'87* (Santa Barbara, CA, USA, Aug. 16–20, 1988), C. Pomerance, Ed., vol. 293 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 392–397.

[129] Lucks, S. The sum of PRPs is a secure PRF. In *Advances in Cryptology – EUROCRYPT 2000* (Bruges, Belgium, May 14–18, 2000), B. Preneel, Ed., vol. 1807 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 470–484.

[130] McEvoy, R. P., Tunstall, M., Murphy, C., and Marnane, W. Differential power analysis of HMAC based on SHA-2, and countermeasures. In *WISA 07: 8th International Workshop on Information Security Applications* (Jeju Island, Korea, Aug. 27–29, 2007), S. Kim, M. Yung, and H.-W. Lee, Eds., vol. 4867 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 317–332.

[131] McGrew, D. An interface and algorithms for authenticated encryption. RFC 5116, Jan. 2008.

[132] McGrew, D., and Fluhrer, S. Multiple forgeries against message authentication codes. Cryptology ePrint Archive, Report 2005/161, May 2005.

[133] McGrew, D., and Viega, J. The security and performance of the Galois/Counter Mode of operation (full version). Cryptology ePrint Archive, Report 2004/193, Aug. 2004.

[134] McGrew, D., and Viega, J. The Galois/Counter Mode of operation (GCM). Submission to NIST, available from their web page, May 2005.

[135] McGrew, D., and Viega, J. GCM update, May 2005. Manuscript, available from NIST's webpage.

[136] McGrew, D., and Viega, J. The Use of Galois Message Authentication Code (GMAC) in IPsec ESP and AH. RFC 4543, May 2006.

[137] McGrew, D. A., and Viega, J. The security and performance of the Galois/counter mode (gcm) of operation. In *Progress in Cryptology - INDOCRYPT 2004: 5th International Conference in Cryptology in India* (Chennai, India, Dec. 20–22, 2004), A. Canteaut and K. Viswanathan, Eds., vol. 3348 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 343–355.

[138] Menezes, A., van Oorschot, P., and Vanstone, S. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, Florida, 1996.

[139] Menezes, A., van Oorschot, P., and Vanstone, S. *Handbook of Applied Cryptography*. The CRC Press series on discrete mathematics and its applications. CRC Press, 2000 N.W. Corporate Blvd., Boca Raton, FL 33431-9868, USA, 1997.

[140] Merkle, R., and Hellman, M. On the security of multiple encryption. *Communications of the ACM 24*, 7 (1981), 465–467.

[141] Minematsu, K. Improved security analysis of XEX and LRW modes. In *SAC 2006: 13th Annual International Workshop on Selected Areas in Cryptography* (Montreal, Canada, Aug. 17–18, 2006), E. Biham and A. M. Youssef, Eds., vol. 4356 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 96–113.

[142] Minematsu, K. Beyond-birthday-bound security based on tweakable block cipher. In *Fast Software Encryption – FSE 2009* (Leuven, Belgium, Feb. 22–25, 2009), O. Dunkelman, Ed., vol. 5665 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 308–326.

[143] Mister, S., and Zuccherato, R. J. An attack on CFB mode encryption as used by OpenPGP. In *SAC 2005: 12th Annual International Workshop on Selected Areas in Cryptography* (Kingston, Ontario, Canada, Aug. 11–12, 2005), B. Preneel and S. Tavares, Eds., vol. 3897 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 82–94.

[144] Mitchell, C. A new key recovery attack on the ANSI retail MAC. *Electronics Letters 39*, 4 (2003).

[145] Mitchell, C. On the security of XCBC, TMAC, and OMAC, Aug. 2003. Technical Report RHUL-MA-2003-4.

[146] Mitchell, C. Error oracle attacks on CBC mode: Is there a future for CBC mode encryption? In *ISC 2005: 8th International Conference on Information Security* (Singapore, Sept. 20–23, 2005), J. Zhou, J. Lopez, R. H. Deng, and F. Bao, Eds., vol. 3650 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 244–258.

[147] Nandi, M. Improved security analysis for OMAC as a pseudornadom function. *Jounral of Mathematical Cryptology 3*, 2 (1998), 133–148.

[148] Nandi, M. A simple and unified method of proving indistinguishability. In *Progress in Cryptology - INDOCRYPT 2006: 7th International Conference in Cryptology in India* (Kolkata, India, Dec. 11–13, 2006), R. Barua and T. Lange, Eds., vol. 4329 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 317–334.

[149] Nandi, M. A unified method for improving PRF bounds for a class of blockcipher based MACs. In *Fast Software Encryption – FSE 2010* (Seoul, Korea, Feb. 7–10, 2010), S. Hong and T. Iwata, Eds., vol. 6147 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 212–229.

[150] Naor, M., and Reingold, O. A pseudo-random encryption mode, 1997. Manuscript, available from Naor's webpage.

[151] Naor, M., and Reingold, O. On the construction of pseudorandom permutations: Luby-Rackoff revisited. *Journal of Cryptology 12*, 1 (1999), 29–66.

[152] Naor, M., and Reingold, O. Synthesizers and their application to the parallel construction of pseudo-random functions. *J. Comput. Syst. Sci. 58*, 2 (1999), 336–375.

[153] National Bureau of Standards. DES modes of operation. Federal Information Processing Standards Publication 81 (FIPS PUB 81), U.S. Department of Commerce, 1980.

[154] National Institute of Standards, and Technology. FIPS PUB 197. Advanced Encryption Standard (AES). U.S. Department of Commerce, Nov. 2001.

[155] National Institute of Standards and Technology. Block ciphers: approved algorithms. Web page maintained by NIST, last visited in January of 2011. http://csrc.nist.gov/groups/ST/toolkit/block_ciphers.html.

[156] National Institute of Standards and Technology. Escrowed Encryption Standard (EES). FIPS PUB 185, U.S. Department of Commerce, Feb. 1994.

[157] National Institute of Standards and Technology. Security requirements for cryptographic modules. FIPS PUB 140-2, U.S. Department of Commerce, May 2001.

[158] National Institute of Standards and Technology. Specification for the Advanced Encryption Standard (AES). FIPS PUB 197, U.S. Department of Commerce, Nov. 2001.

[159] National Institute of Standards and Technology. FIPS PUB 198-1. The Keyed-Hash Message Authentication Code (HMAC). Federal Information Processing Standards Publication, July 2009.

[160] National Security Agency. SKIPJACK and KEA algorithm specifications. Version 2.0. Available from NIST's webpage, May 1998.

[161] Okeya, K. Side channel attacks against HMACs based on block-cipher based hash functions. In *ACISP 06: 11th Australasian Conference on Information Security and Privacy* (Melbourne, Australia, July 3–5, 2006), L. M. Batten and R. Safavi-Naini, Eds., vol. 4058 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 432–443.

[162] OpenSSL. The open source toolkit for SSL/TLS. http://www.openssl.org/source, 2010.

[163] Patarin, J. Security of random Feistel schemes with 5 or more rounds. In *Advances in Cryptology – CRYPTO 2004* (Santa Barbara, CA, USA, Aug. 15–19, 2004), M. Franklin, Ed., vol. 3152 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 106–122.

[164] Paterson, K., and Watson, G. Immunising CBC mode against padding oracle attacks: A formal security treatment. In *SCN 08: 6th International Conference on Security in Communication Networks* (Amalfi, Italy, Sept. 10–12, 2008), R. Ostrovsky, R. D. Prisco, and I. Visconti, Eds., vol. 5229 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 340–357.

[165] Paterson, K., and Yau, A. Padding oracle attacks on the ISO CBC mode encryption standard. In *Topics in Cryptology – CT-RSA 2004* (San Francisco, CA, USA, Feb. 23–27, 2004), T. Okamoto, Ed., vol. 2964 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 305–323.

[166] Paterson, K., and Yau, A. Cryptography in theory and practice: The case of encryption in ipsec. In *Advances in Cryptology – EUROCRYPT 2006* (St. Petersburg, Russia, May 28 – June 1, 2006), S. Vaudenay, Ed., vol. 4004 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 12–29.

[167] Petrank, E., and Rackoff, C. CBC MAC for real-time data sources. *Journal of Cryptology 13*, 3 (2000), 315–338.

[168] Pietrzak, K. A tight bound for EMAC. In *ICALP 2006: 33rd International Colloquium on Automata, Languages and Programming, Part II* (Venice, Italy, July 10–14, 2006), M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, Eds., vol. 4052 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 168–179.

[169] Preneel, B. MAC algorithms: State of the art and recent developments, May 2008. Lecture slides, available from the speaker's personal webpage.

[170] Preneel, B., and van Oorschot, P. Key recovery attack on ANSI X9.19 retail MAC. *Electronics Letters 32*, 17 (1996), 1568–1569.

[171] Preneel, B., and van Oorschot, P. On the security of iterated message authentication codes. *IEEE Transactions on Information Theory 45*, 1 (1999), 188–199.

[172] Preneel, B., and van Oorschot, P. C. MDx-MAC and building fast MACs from hash functions. In *Advances in Cryptology – CRYPTO'95* (Santa Barbara, CA, USA, Aug. 27–31, 1995), D. Coppersmith, Ed., vol. 963 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 1–14.

[173] Rechberger, C., and Rijmen, V. On authentication with HMAC and non-random properties. In *FC 2007: 11th International Conference on Financial Cryptography and Data Security* (Scarborough, Trinidad and Tobago, Feb. 12–16, 2007), S. Dietrich and R. Dhamija, Eds., vol. 4886 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 119–133.

[174] Rhroeppel, R. Public comments on the XTS-AES mode. Collected email comments released by NIST, available from their web page, 2008.

[175] Ristenpart, T., and Rogaway, P. How to enrich the message space of a cipher. In *Fast Software Encryption – FSE 2007* (Luxembourg, Luxembourg, Mar. 26–28, 2007), A. Biryukov, Ed., vol. 4593 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 101–118.

[176] Rogaway, P. Problems with proposed IP cryptography. Manuscript draft-rogaway-ipsec-comments-00.txt, Apr. 1995.

[177] Rogaway, P. Bucket hashing and its application to fast message authentication. *Journal of Cryptology 12*, 2 (1999), 91–115.

[178] Rogaway, P. Authenticated-encryption with associated-data. In *ACM CCS 02: 9th Conference on Computer and Communications Security* (Washington D.C., USA, Nov. 18–22, 2002), V. Atluri, Ed., ACM Press, pp. 98–107.

[179] Rogaway, P. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In *Advances in Cryptology – ASIACRYPT 2004* (Jeju Island, Korea, Dec. 5–9, 2004), P. J. Lee, Ed., vol. 3329 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 16–31.

[180] Rogaway, P. Nonce-based symmetric encryption. In *Fast Software Encryption – FSE 2004* (New Delhi, India, Feb. 5–7, 2004), B. Roy and W. Meier, Eds., vol. 3017 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 348–359.

[181] Rogaway, P. Public comments on the XTS-AES mode. Collected email comments released by NIST, available from their web page, 2008.

[182] Rogaway, P., Bellare, M., Black, J., and Krovetz, T. OCB: A block-cipher mode of operation for efficient authenticated encryption. In *ACM CCS 01: 8th Conference on Computer and Communications Security* (Philadelphia, PA, USA, Nov. 5–8, 2001), ACM Press, pp. 196–205.

[183] Rogaway, P., and Shrimpton, T. A provable-security treatment of the key-wrap problem. In *Advances in Cryptology – EUROCRYPT 2006* (St. Petersburg, Russia, May 28 – June 1, 2006), S. Vaudenay, Ed., vol. 4004 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 373–390.

[184] Rogaway, P., and Wagner, D. A critique of CCM. Cryptology ePrint Archive, Report 2003/070, Apr. 2003.

[185] Rogway, P., and Zhang, H. Online ciphers from tweakable blockciphers. In *Topics in Cryptology – CT-RSA 2011* (2011), Lecture Notes in Computer Science, Springer, Berlin, Germany.

[186] Salowey, J., Choudhury, A., and McGrew, D. AES Galois Counter Mode (GCM) Cipher Suites for TLS. RFC 5288, Aug. 2008.

[187] Schroeppel, R. The Hasty Pudding Cipher. AES candidate submitted to NIST. http://www.cs.arizona/∼rcs/hpc, 1998.

[188] Shannon, C. Communication theory of secrecy systems. *Bell Systems Technical Journal 28*, 4 (1949), 656–715.

[189] Simon, D. Finding collisions on a one-way street: Can secure hash functions be based on general assumptions? In *Advances in Cryptology – EUROCRYPT'98* (Espoo, Finland, May 31 – June 4, 1998), K. Nyberg, Ed., vol. 1403 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 334–345.

[190] Stevens, M., Lenstra, A., and de Weger, B. Chosen-prefix collisions for MD5 and colliding X.509 certificates for different identities. In *Advances in Cryptology – EUROCRYPT 2007* (Barcelona, Spain, May 20–24, 2007), M. Naor, Ed., vol. 4515 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 1–22.

[191] Stevens, M., Sotirov, A., Appelbaum, J., Lenstra, A., Molnar, D., Osvik, D., and de Weger, B. Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate. In *Advances in Cryptology – CRYPTO 2009* (Santa Barbara, CA, USA, Aug. 16–20, 2009), S. Halevi, Ed., vol. 5677 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 55–69.

[192] Stubblebine, S., and Gligor, V. On message integrity in cryptographic protocols. In *IEEE Computer Symposium on Research in Security and Privacy* (1992), IEEE Press, pp. 85–104.

[193] Tsudik, G. Message authentication with one-way hash functions. In *INFOCOM* (1992), pp. 2055–2059.

[194] Vaudenay, S. Security flaws induced by CBC padding — Applications to SSL, IPSEC, WTLS .... In *Advances in Cryptology – EUROCRYPT 2002* (Amsterdam, The Netherlands, Apr. 28 – May 2, 2002), L. R. Knudsen, Ed., vol. 2332 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 534–546.

[195] Viega, J., and McGrew, D. The use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP). RFC 4106, June 2005.

[196] Wang, L., Ohta, K., and Kunihiro, N. New key-recovery attacks on HMAC/NMAC-MD4 and NMAC-MD5. In *Advances in Cryptology – EUROCRYPT 2008* (Istanbul, Turkey, Apr. 13–17, 2008), N. P. Smart, Ed., vol. 4965 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 237–253.

[197] Wang, P., Feng, D., Lin, C., and Wu, W. Security of truncated MACs. In *Inscrypt* (2008), M. Yung, P. Liu, and D. Lin, Eds., vol. 5487 of *Lecture Notes in Computer Science*, Springer, pp. 96–114.

[198] Wang, X., Yin, Y., and Yu, H. Finding collisions in the full SHA-1. In *Advances in Cryptology – CRYPTO 2005* (Santa Barbara, CA, USA, Aug. 14–18, 2005), V. Shoup, Ed., vol. 3621 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 17–36.

[199] Wang, X., and Yu, H. How to break MD5 and other hash functions. In *Advances in Cryptology – EUROCRYPT 2005* (Aarhus, Denmark, May 22–26, 2005), R. Cramer, Ed., vol. 3494 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 19–35.

[200] Wang, X., Yu, H., Wang, W., Zhang, H., and Zhan, T. Cryptanalysis on HMAC/NMAC-MD5 and MD5-MAC. In *Advances in Cryptology – EUROCRYPT 2009* (Cologne, Germany, Apr. 26–30, 2009), A. Joux, Ed., vol. 5479 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 121–133.

[201] Wegman, M., and Carter, L. New hash functions and their use in authentication and set equality. In *J. of Comp. and System Sciences* (1981), vol. 22, pp. 265–279.

[202] Wen, F., Wu, W., and Wen, Q. Error oracle attacks on several modes of operation. In *CIS (2)* (2005), Y. Hao, J. Liu, Y. Wang, Y. ming Cheung, H. Yin, L. Jiao, J. Ma, and Y.-C. Jiao, Eds., vol. 3802 of *Lecture Notes in Computer Science*, Springer, pp. 62–67.

[203] Whiting, D., Housley, R., and Ferguson, N. Counter with CBC-MAC (CCM). Undated manuscript. Submission to NIST, available from their web page, June 2002.

[204] Whiting, D., Housley, R., and Ferguson, N. Counter with CBC-MAC (CCM). RFC 3610 (Informational), Sept. 2003.

[205] Wikipedia. Disk encryption theory. Wikipedia encyclopedia entry. Visited January 9, 2011.

[206] Yasuda, K. The sum of CBC MACs is a secure PRF. In *Topics in Cryptology – CT-RSA 2010* (San Francisco, CA, USA, Mar. 1–5, 2010), J. Pieprzyk, Ed., vol. 5985 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 366–381.

[207] Yau, A., Paterson, K., and Mitchell, C. Padding oracle attacks on CBC-mode encryption with secret and random IVs. In *Fast Software Encryption – FSE 2005* (Paris, France, Feb. 21–23, 2005), H. Gilbert and H. Handschuh, Eds., vol. 3557 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, pp. 299–319.

[208] Zhou, G., Michalik, H., and Hinsenkamp, L. Improving throughput of AES-GCM with pipelined Karatsuba multipliers on FPGAs. In *ARC* (2009), J. Becker, R. Woods, P. M. Athanas, and F. Morgan, Eds., vol. 5453 of *Lecture Notes in Computer Science*, Springer, pp. 193–203.