UNIVERSITY OF CALIFORNIA, SAN DIEGO

Authenticated Encryption in Practice: Generalized Composition Methods and the Secure Shell, CWC, and WinZip Schemes

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in
Computer Science

by

Tadayoshi Kohno

Committee in charge:

Professor Mihir Bellare, Chair
Professor Rene Cruz
Professor Bill Lin
Professor Daniele Micciancio
Professor Stefan Savage

2006

The dissertation of Tadayoshi Kohno is approved, and it is acceptable in quality and form for publication on microfilm:

_____

_____

_____

_____

_____
Chair

University of California, San Diego

2006

To Taryn and Seth Kohno.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

ACADEMIC COLLABORATORS AND FRIENDS. I thank my fantastic advisor, Mihir Bellare, for all the wonderful advice and guidance that he has given me over the years. I truly believe that I would not be who I am now if it were not for him.

I am indebted to Stefan Savage for all of his generosity and guidance and for helping me find my "dream job." I thank Avi Rubin for his continual mentoring and constant generosity both professionally and socially. I thank Sid Karin and Dan Wallach for all their help and advice and for watching over my graduate career.

I thank my summer mentors and past supervisors, kc claffy, David Conrad, Mark McGovern, Gary McGraw, Fabian Monrose, Bruce Schneier, David Wagner, Tammy Welcome, and Phil Winterbottom. I thank Rene Cruz, Bill Lin, and Daniele Micciancio for overseeing this dissertation. I thank Hal Gabow and Evi Nemeth for overseeing my undergraduate career.

In addition to Mihir Bellare, I thank John Black, Chanathip Namprempre, Adriana Palacio, John Viega, and Doug Whiting for co-authoring with me some of the material that appears in this dissertation. I also thank all my other co-authors and collaborators, Michel Abdalla, J. T. Bloch, Andre Broido, Dario Catalano, Niels Ferguson, Kevin Fu, Chris Hall, Tetsu Iwata, Seny Kamara, John Kelsey, Eike Kiltz, Lars Knudsen, Tanja Lange, Stefan Lucks, John Malone-Lee, David Molnar, Gregory Neven, Pascal Paillier, Bruce Potter, Naveen Sastry, Haixia Shi, Mike Stay, and Adam Stubblefield.

I thank Emile Aben, Dan Andersen, Matt Bishop, Alexandra Boldyreva, Dan Brown, Cindy Cohn, Don Coppersmith, Frank Dabek, David Dill, Morris Dworkin, Hal Finney, Michael Freedman, Beth Friedman, Patricia Gabow, Brian Gladman, Philippe Golle, Bill Griswold, Peter Gutmann, Stuart Haber, Susan Hohenberger, Tim Hollebeek, Young Hyun, Russell Impagliazzo, David Jefferson, Rob Johnson, Frans Kaashoek, Chris Karlof, Ulrich Kuehn, Mahesh Kallahalla, David Mazières, David McGrew, David Moore, Badri Natarajan, Bart Preneel, Christian Rechberger, Mike Reiter, Ron Rivest, Phil Rogaway, Felix Schleer, Rich Schroeppel, Jason Schultz, Umesh Shankar, Colleen Shannon, abhi shelat, Tsutomu Shimomura, Emil Sit, Nigel Smart, Bill Sommerfeld,

Alex Snoeren, Jeremy Stribling, Ram Swaminathan, Win Treese, Darryl Veitch, Tracy Volz, Brendan White, Richard Wiebe, Michael Wiener, Pat Wilson, Matt Zimmerman, and Robert Zuccherato for commenting on my papers and contributing to my research. I thank Phil Zimmermann for introducing me to modern applied cryptography. I additionally thank Josh Benaloh, Brad Calder, Trent Jaeger, Patrick McDaniel, Dave Schroeder, Dan Simon, Hiroyuki Tanabe, Geoff Voelker, and Bennet Yee for all their contributions to my career. I thank all the other members of the UCSD cryptography and security group, including Jee Hea An, Marc Fischlin, Alejandro Hevia, Matt Hohlfeld, Anton Mityagin, Saurabh Panjwani, Barath Raghavan, Tom Ristenpart, Sarah Shoup, Bogdan Warinschi, and Scott Yilek. I am grateful for all the help from Julie Conner and Kathy Reed and rest of the UCSD CSE staff, and Steve Hopper and the rest CSEHelp. I thank everyone else that I have had contact with academically and socially.

FAMILY. I am deeply grateful for all that my wife Taryn and son Seth have contributed to all aspects of my life. I could mention a few things, like Taryn always putting my education and career before herself, but I honestly do not believe that any summary of their contributions will do them justice. Indeed, in any short summary I would be forced to exclude many of their critical contributions and sacrifices.

I have always had an interest in the maths and sciences, and for that I thank my family, and in particular my parents, Tadahiko and Beth. I thank my parents for also endlessly investing in my education, including the many computers and electronic equipment that they purchased for me as a child and the many hours that they spent shuttling me back and forth between Fairview and CU.

FUNDING AND SUPPORT. My graduate research was supported in part by a National Defense Science and Engineering Graduate Fellowship, an IBM Ph.D. Fellowship, the SciDAC program of the US Department of Energy (award DE-FC02-01ER25466), and NSF grants ANR-0129617, CCR-0093337, and CCR-0208842. I thank the Usenix Association for a Student Grant supporting my work on SSH. I thank The Johns Hopkins University Information Security Institute (Avi Rubin and Fabian Monrose) for hosting me the summer of 2003, the Cooperative Association for Internet Data Analysis (kc

claffy) for hosting me the summer of 2004, and the University of California at Berkeley (David Wagner) for hosting me the summer of 2005.

PAPERS INCLUDED IN THIS DISSERTATION. An earlier version of the material in Chapter 3 appears in the ACM Transactions on Information and System Security [8], copyright the ACM. I was a primary researcher for this work. The full citation for this work is:

> Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the Encode-then-Encrypt-and-MAC paradigm. *ACM Transactions on Information and System Security*, 7(2):206–241, May 2004.

The material in Chapter 4 comes from in-progress work. I was a primary researcher for this work, the full citation of which is currently:

> Tadayoshi Kohno, Adriana Palacio, and John Black. Authenticated-encryption: New notions and constructions. Manuscript, 2006.

An earlier version of the material in Chapter 5 appears in Fast Software Encryption, volume 3017 of Lecture Notes in Computer Science [50], copyright the IACR. I was a primary researcher for the theoretical results in this paper. The full citation for this work is:

> Tadayoshi Kohno, John Viega, and Doug Whiting. CWC: A high-performance conventional authenticated encryption mode. In Bimal Roy and Willi Meier, editors, *Fast Software Encryption*, volume 3017 of *Lecture Notes in Computer Science*, pages 408–426. Springer-Verlag, February 2004.

An earlier version of the material in Chapter 6 appears in the Proceedings of the 11th ACM Conference on Computer and Communications Security [49], copyright the ACM. I was a primary researcher and single-author on this paper. The full citation for this work is:

Tadayoshi Kohno. Attacking and repairing the WinZip encryption scheme. In Birgit Pfitzmann, editor, *Proceedings of the 11th ACM Conference on Computer and Communications Security*, pages 72–81. ACM Press, October 2004.

VITA

1999 B.S. University of Colorado, Boulder

2004 M.S. University of California, San Diego

2006 Ph.D. University of California, San Diego

PUBLICATIONS

Harold N. Gabow and Tadayoshi Kohno. A network-flow-based scheduler: Design, performance history, and experimental analysis. In *Second Workshop on Algorithm Engineering and Experiment*, pages 1–14, January 2000.

John Kelsey, Tadayoshi Kohno, and Bruce Schneier. Amplified boomerang attacks against reduced-round MARS and Serpent. In Bruce Schneier, editor, *Fast Software Encryption*, volume 1978 of *Lecture Notes in Computer Science*, pages 75–93. Springer-Verlag, April 2000.

Tadayoshi Kohno, John Kelsey, and Bruce Schneier. Preliminary cryptanalysis of reduced-round Serpent. In *Third AES Candidate Conference*, pages 195–211, April 2000.

John Viega, J. T. Bloch, Yoshi Kohno, and Gary McGraw. ITS4: A static vulnerability scanner for C and C++ code. In *Sixteenth Annual Computer Security Applications Conference*, pages 257–267, December 2000.

Harold N. Gabow and Tadayoshi Kohno. A network-flow-based scheduler: Design, performance history, and experimental analysis. *ACM Journal of Experimental Algorithmics*, 6, 2001.

Tadayoshi Kohno and Mark McGovern. On the global content PMI: Improved copy-protected Internet content distribution. In Paul F. Syverson, editor, *Financial Cryptography*, volume 2339 of *Lecture Notes in Computer Science*, pages 79–90. Springer-Verlag, February 2001.

John Viega, Tadayoshi Kohno, and Bruce Potter. Trust (and mistrust) in secure applications. *Communications of the ACM*, 44(2):31–36, February 2001.

John Viega, J. T. Bloch, Tadayoshi Kohno, and Gary McGraw. Token-based scanning for source code security problems. *ACM Transactions on Information and System Security*, 5(3):238–261, August 2002.

Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. Authenticated encryption in SSH: Provably fixing the SSH binary packet protocol. In Vijay Atluri, editor, *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 1–11. ACM Press, November 2002.

Niels Ferguson, Doug Whiting, Bruce Schneier, John Kelsey, Stefan Lucks, and Tadayoshi Kohno. Helix: Fast encryption and authentication in a single cryptographic primitive. In Thomas Johansson, editor, *Fast Software Encryption*, volume 2887 of *Lecture Notes in Computer Science*, pages 330–346. Springer-Verlag, February 2003.

Lars R. Knudsen and Tadayoshi Kohno. Analysis of RMAC. In Thomas Johansson, editor, *Fast Software Encryption*, volume 2887 of *Lecture Notes in Computer Science*, pages 182–191. Springer-Verlag, February 2003.

Mihir Bellare and Tadayoshi Kohno. A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 491–506. Springer-Verlag, May 2003.

Tadayoshi Kohno, John Viega, and Doug Whiting. CWC: A high-performance conventional authenticated encryption mode. In Bimal Roy and Willi Meier, editors, *Fast Software Encryption*, volume 3017 of *Lecture Notes in Computer Science*, pages 408–426. Springer-Verlag, February 2004.

Tetsu Iwata and Tadayoshi Kohno. New security proofs for the 3GPP confidentiality and integrity algorithms. In Bimal Roy and Willi Meier, editors, *Fast Software Encryption*, volume 3017 of *Lecture Notes in Computer Science*, pages 427–445. Springer-Verlag, February 2004.

Mihir Bellare and Tadayoshi Kohno. Hash function balance and its impact on birthday attacks. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 401–418. Springer-Verlag, May 2004.

Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin, and Dan S. Wallach. Analysis of an electronic voting system. In *IEEE Symposium on Security and Privacy*, pages 27–40. IEEE Computer Society, May 2004.

Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the Encode-then-Encrypt-and-MAC paradigm. *ACM Transactions on Information and System Security*, 7(2):206–241, May 2004.

Tadayoshi Kohno. Attacking and repairing the WinZip encryption scheme. In Birgit Pfitzmann, editor, *Proceedings of the 11th ACM Conference on Computer and Communications Security*, pages 72–81. ACM Press, October 2004.

Tadayoshi Kohno, Andre Broido, and kc claffy. Remote physical device fingerprinting. In *IEEE Symposium on Security and Privacy*, pages 211–225. IEEE Computer Society, May 2005.

Tadayoshi Kohno, Andre Broido, and K.C. Claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2(2):93–108, April–June 2005.

Michel Abdalla, Mihir Bellare, Dario Catalano, Eike Kiltz, Tadayoshi Kohno, Tanja Lange, John Malone-Lee, Gregory Neven, Pascal Paillier, and Haixia Shi. Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 205–222. Springer-Verlag, August 2005.

Kevin Fu, Seny Kamara, and Tadayoshi Kohno. Key regression: Enabling efficient key distribution for secure distributed storage. In *ISOC Network and Distributed System Security Symposium*, February 2006.

David Molnar, Tadayoshi Kohno, Naveen Sastry, and David Wagner. Tamper-evident, history-independent, subliminal-free data structures on PROM storage -or- how to store ballots on a voting machine (extended abstract). In *IEEE Symposium on Security and Privacy*. IEEE Computer Society, May 2006.

John Kelsey and Tadayoshi Kohno. Herding hash functions and the Nostradamus attack. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, Lecture Notes in Computer Science. Springer-Verlag, May 2006.

Naveen Sastry, Tadayoshi Kohno, and David Wagner. Designing voting machines for verification. In *15th Usenix Security Symposium*. Usenix, August 2006.

ABSTRACT OF THE DISSERTATION

Authenticated Encryption in Practice: Generalized Composition Methods and the Secure Shell, CWC, and WinZip Schemes

by

Tadayoshi Kohno
Doctor of Philosophy in Computer Science
University of California, San Diego, 2006
Professor Mihir Bellare, Chair

We study authenticated encryption (AE) schemes, or symmetric cryptographic protocols designed to protect both the privacy and the integrity of digital communications. When the AE schemes that we propose or study are secure, we prove so using the modern cryptography approach of practice-oriented provable security; this approach involves formally defining what it means for an AE scheme to be secure, and then deriving proofs of security via reductions from the security of the construction's underlying components. When we find that an AE scheme is insecure, we support our discoveries with example attacks and then propose security improvements.

We first study the AE portion of the Secure Shell (SSH) protocol. The SSH AE scheme is based on the Encrypt-and-MAC paradigm. Despite previous negative results on the Encrypt-and-MAC paradigm, we prove that the overall design of the SSH AE scheme is secure under reasonable assumptions. Our proofs for SSH contribute to the field of cryptography in several ways. First, we extend previous formal definitions of security for AE schemes to capture additional security goals, namely resistance to replay and re-ordering attacks. We also formalize a new AE paradigm, Encode-then-E&M, that captures the differences between the real SSH AE scheme and the previous Encrypt-and-MAC model. We state provable security results about both the Encode-then-E&M paradigm and the SSH AE scheme.

Motivated by the differences between previous models and real AE schemes, we then consider and prove security results about generalizations of two other natural AE paradigms, MAC-then-Encrypt and Encrypt-then-MAC, as well as further generalizations of the Encode-then-E&M paradigm. Motivated by practical requirements and the IPsec community, we propose CWC — the first block cipher-based AE scheme that is simultaneously provably secure, fully parallelizable, and free from intellectual property claims. Finally, we discover and propose fixes to security defects with the WinZip AE-2 AE scheme. Our attacks exploit interactions between AE-2's provably secure Encrypt-then-MAC core and the rest of the system. Since WinZip could have avoided certain attacks by applying the provable security approach to the whole AE-2 scheme, our results suggest the importance of pushing the provable security approach further into real systems.

1 Introduction

In this dissertation we consider two of the most fundamental goals in cryptography: the protection of the *privacy* of digital communication and the protection of the *integrity* of digital communication. We consider these goals in the *symmetric* or *shared-key* setting, which means that we are interested in studying privacy- and integrity-preserving mechanisms when two communicating parties already share some secret information, called a cryptographic *key*.

1.1 Authenticated Encryption

While cryptographers have long realized the importance of the *privacy* and *integrity* goals, the traditional approach in cryptography research has been to consider these two goals in isolation. In this vein, cryptographers designed *encryption schemes* [4, 30] to protect the privacy of digital communication and they designed *message authentication schemes* (a.k.a. *MACs*) [6, 30] to protect the integrity of digital communication. It was not until recently that cryptographers began to formally study *authenticated encryption schemes*, or mechanisms for achieving both of these goals *simultaneously* [10, 11, 47, 52].

Some of the initial results on authenticated encryption might appear counter-intuitive. For example, Bellare and Namprempre [10] and Krawczyk [52] both proved that one natural method for combining a secure, traditional, privacy-only encryption

scheme with a secure, traditional, integrity-only MAC will *always* fail to provide privacy when the MAC is stateless and deterministic, as many popular MACs are. As a concrete example, using this approach to combine the popular AES-CTR encryption scheme with the popular HMAC-SHA1 message authentication scheme will result in an *insecure* (not privacy-preserving) authenticated encryption scheme. The authors dub this natural but insecure method for combining an encryption scheme with a MAC the "Encrypt-*and*-MAC" paradigm.

Results like those of Bellare-Namprempre and Krawczyk suggest a need for further investigation into the theory of how to create authenticated encryption schemes with strong theoretical support, i.e., how to create authenticated encryption schemes that are *provably secure* [6, 37] under reasonable assumptions. Toward this end, Bellare-Namprempre and Krawczyk also show that another natural method of combining an encryption scheme and a MAC does yield a composite authenticated encryption scheme that provably provides both strong privacy and integrity properties at the same time. These authors dub the provably secure composition method the "Encrypt-*then*-MAC" paradigm. (Bellare-Namprempre and Krawczyk additionally consider a third composition method, MAC-*then*-Encrypt, that informally provides a level of security somewhere between that of the Encrypt-and-MAC paradigm and the Encrypt-then-MAC paradigm.)

Since many popular encryption schemes and MACs are themselves built from one of cryptography's most basic components, the *block cipher*, cryptographers also began to consider how to design authenticated encryption schemes directly from block ciphers. Efficiency was one of the principle motivations for this line of research, and early examples of this research direction include the works of Katz and Yung [47], Jutla [44], Gligor and Donescu [35], and Rogaway, Bellare, and Black [72].

While these early works make great strides in the formal study of authenticated encryption, there remain gaps between the theoretical results on authenticated encryption and the needs of practitioners. For example, the previous theoretical analyses of composition-based authenticated encryption schemes do not fully model many real constructions, like the construction used in the popular Secure Shell (SSH) protocol. Ad-

ditionally, no pre-existing block cipher-based authenticated encryption scheme has all three of the following properties: provable security, full data parallelizability, and freedom from intellectual property claims. Consequently, none of the pre-existing block cipher-based authenticated encryption schemes are suitable for some real deployment scenarios, like high-speed IPsec routers that must handle data at 10 Gbps. (To better understand this latter conclusion, we note that the IETF standardization body dislikes patented constructions, which eliminates the use of parallelizable but patented constructions. On the other hand, unpatented and non-parallelizable constructions cannot achieve 10 Gbps using conventional ASIC technology.) The purpose of this dissertation is to help address these gaps. We summarize our contributions below, after first elaborating on what we mean by *provable security*.

1.2 Provable Security

Goldwasser and Micali [37] introduced the notion of *provable security*, which is based on the science of complexity theory. In the Goldwasser-Micali approach, individuals design a cryptographic scheme based on some believed-to-be computationally hard problems, which they treat as *basic building blocks* or *primitives*. The designers also determine what it means for the scheme to be "secure" by establishing precise *security definitions* that capture the designers' security goals. After determining the appropriate security definitions, the designers (attempt to) prove the security of their construction via a *reduction* from the hardness of the underlying building blocks, similar to the way one reduces SAT to a problem to prove that the problem is NP-hard. The use of reductions allows the designers to prove that *any* efficient attack against the security of their scheme would directly correspond to an efficient solution to the problem posed by one of the underlying building blocks, and as long as the building blocks are truly hard, we know that the designers' scheme is secure under the chosen security definitions.

Bellare, Kilian, and Rogaway [6] built on the line of work that Goldwasser and Micali started and, in doing so, created the field of *practice-oriented provable security*. The

fundamental difference between BKR's approach and Goldwasser-Micali's approach is that BKR were interested in the design and analysis of practical, implemented protocols and performed their analyses using concrete reductions from finite objects, whereas Goldwasser-Micali were largely interested in complexity theoretic questions concerning the asymptotic relationships between different security goals. Since its conception, many researchers have extensively applied the practice-oriented provable security approach in the design and analysis of practical cryptosystems. In the case of symmetric schemes, the underlying basic building blocks are generally engineered objects, such as the *block cipher* AES [28].

1.3 The Secure Shell Protocol and Composition-Based Authenticated Encryption Schemes

In this dissertation we revisit the composition methods that Bellare and Namprempre [10] and Krawczyk [52] analyzed in their early papers on authenticated encryption. Specifically, we begin by looking at the authenticated encryption portion of the popular Secure Shell (SSH) protocol. Of particular interest here is that the SSH authenticated encryption scheme is based on the insecure Encrypt-and-MAC paradigm, yet we are able to show that the SSH approach for combining an encryption scheme and a MAC is in fact secure under reasonable assumptions.

Our provable security results do not contradict the results of Bellare-Namprempre and Krawczyk. Rather, the critical property that we exploit in our proofs of security is that, while the SSH protocol is based on the insecure Encrypt-and-MAC paradigm, it has a few slight differences. Namely, the SSH protocol preprocesses user data (the data that we are concerned about from a privacy and integrity perspective) before invoking the underlying encryption scheme and MAC, thereby taking the SSH construction outside of the generic Encrypt-and-MAC model and making Bellare-Namprempre's and Krawczyk's results inapplicable. We generalize our analysis by modeling the preprocessing step as an *encoding scheme*, and then prove general results for a new paradigm

that we call *Encode-then-Encrypt-and-MAC* (*Encode-then-E&M*). Our results on the SSH protocol are in Chapter 3.

Motivated by our SSH results, and by the fact that real protocols seldom employ any of the basic Encrypt-and-MAC, Encrypt-then-MAC, and MAC-then-Encrypt paradigms without alteration, we then formally study new abstractions that we call the *generalized Encode-then-E&M*, the *generalized Encode-then-EtM*, and the *generalized Encode-then-MtE* paradigms. We present these results in Chapter 4. The generalized Encode-then-E&M paradigm that we consider in Chapter 4 is a more flexible version of the Encode-then-E&M paradigm that we consider in Chapter 3.

For our provable security results in Chapters 3 and 4, we also introduce new, strong, formal definitions of privacy and integrity for authenticated encryption schemes. Our definitions extend the standard definitions of privacy and integrity [4, 10, 11, 47], but also capture additional security goals that developers often desire. For example, if a construction is provably secure under one of our new definitions of security, then that construction will provably resist *replay* and *out-of-order delivery* attacks.

As an aside, our analysis of the SSH authenticated encryption scheme did uncover a privacy vulnerability, but this vulnerability is not endemic of the high level SSH construction. Rather, the problem that we identify stems from a poor choice for SSH's underlying encryption scheme. Details in Chapter 3.

1.4 The CWC Authenticated Encryption Scheme

In addition to studying composition-based authenticated encryption schemes, we also propose a new block cipher-based authenticated encryption scheme, which we call CWC. CWC is the first block cipher-based authenticated encryption scheme that is simultaneously provably secure, fully parallelizable, and unencumbered by intellectual property issues. One of our pragmatic goals was to provide the first provable secure authenticated encryption scheme for high-speed 10 Gbps IPsec routers.

The heart of the CWC design is to combine a Carter-Wegman-style polynomial

universal hash function-based message authentication scheme [81] with a counter (CTR) mode encryption scheme, but to do so in an invasive, non-generic way. For example, while it is generally a poor security design decision to use the same cryptographic key in an encryption scheme and a MAC, we design CWC in such a way that one can use the same block cipher key in both CWC's underlying encryption component and CWC's underlying message authentication component. Sharing a key in this manner is advantageous since it minimizes expensive memory accesses in high-speed hardware. Details in Chapter 5.

1.5 The WinZip Authenticated Encryption Scheme

In Chapter 6 we cryptanalyze WinZip Computing, Inc.'s new AE-2 authenticated encryption scheme. (WinZip Computing, Inc. is the creator of the popular WinZip file utility program for Windows machines, as well as an Outlook email plugin.) Unlike previous chapters, Chapter 6 does not contain provable security results, but rather serves to help underscore the importance of extending provable security further into real systems by highlighting examples of security issues that can arise when a construction is not provably secure, or when a larger system uses a provably secure sub-component without fully addressing the security of the connection between that sub-component and the larger system.

In more detail, the core of the WinZip AE-2 authenticated encryption scheme is a provably secure Encrypt-then-MAC construction, where the underlying encryption component is the popular AES-CTR mode encryption scheme and the underlying authentication component is the popular HMAC-SHA1 message authentication scheme. Our attacks do not invalidate the security of the AE-2 Encrypt-then-MAC core, but rather exploit problems with the interface between the Encrypt-then-MAC core and the rest of the WinZip system. For example, one of our attacks exploits the way that WinZip preprocesses and compresses user data before processing that data with the Encrypt-then-MAC core. We also uncover a chosen-ciphertext attack that exploits Windows'

association of applications (e.g., Microsoft Word) to filename extensions (e.g., `.doc`) and the fact that WinZip does not cryptographically protect the integrity of an encapsulated file's filename. We further uncover security issues that arise because of the fact that when a WinZip archive contains multiple files, each file is compressed and encrypted independently. Details of these and other results, as well as our recommended fixes, are in Chapter 6.

2 Background

2.1 Notation

If x and y are strings, then $|x|$ denotes the length of x in bits and $x\|y$ denotes their concatenation. If i is a non-negative integer and l is a positive integer, $0 \leq i < 2^l$, then $\langle i \rangle_l$ denotes the unsigned l-bit binary representation of i in big-endian format. If x is a string, then $\mathsf{toint}(x)$ denotes the integer corresponding to string x in big-endian format (the most significant bit is *not* interpreted as a sign bit). For example, $\mathsf{toint}(10000010) = 2^7 + 2 = 130$. If b is a bit and n a non-negative integer, then b^n denote b concatenated with itself n times; e.g., 10^7 is the string 10000000. If $a_1, \ldots, a_m$ are strings, then $\langle a_1, \ldots, a_m \rangle$ denotes an injective encoding from the set of all possible values for $a_1, \ldots, a_m$, which will be clear from context, to a set of strings such that $a_1, \ldots, a_m$ are recoverable. We denote the empty string by ε. When we say an algorithm is stateful, we mean that it uses and updates its state and that the entity executing it maintains the state between invocations. Let ε denote the initial state of any (stateful or stateless) algorithm. Let $x \leftarrow y$ denote the assignment of y to x. If X is a set, then $x \xleftarrow{\$} X$ denotes the process of selecting an element uniformly at random from X and assigning the result to x. If f is a randomized (resp., deterministic) algorithm, then $x \xleftarrow{\$} f(y)$ (resp., $x \leftarrow f(y)$) denotes the process of running f on input y and assigning the result to x. If A is a program, $A \Leftarrow x$ means "return the value x to A." When we refer to the time of an algorithm or experiment, we include the size of the code in some fixed encoding. There is also an implicit big-$\mathcal{O}$ surrounding all such time references.

2.2 Pseudorandom Functions

We formalize the notion of pseudorandom functions following the papers [6, 36]. Let $\mathcal{F}\colon\ \mathcal{K} \times \mathcal{M} \to \mathcal{R}$ be a family of functions from $\mathcal{M}$ to $\mathcal{R}$ indexed by keys $\mathcal{K}$. We use $\mathcal{F}_K(M)$ as shorthand for $\mathcal{F}(K, M)$. Let Rand$[\mathcal{M}, \mathcal{R}]$ denote the set of all functions from $\mathcal{M}$ to $\mathcal{R}$. If l and L are positive integers, we use Rand$[l, L]$ as shorthand for Rand$[\{0,1\}^l, \{0,1\}^L]$. Informally, $\mathcal{F}$ is a *secure pseudorandom function* (PRF) if it is hard for all distinguishers (adversaries) D_{prf} using reasonable resources to distinguish $\mathcal{F}_K(\cdot)$, with a randomly selected key $K \in \mathcal{K}$, from a randomly selected function f from Rand$[\mathcal{M}, \mathcal{R}]$. We make this more formal below.

Definition 2.2.1 (Pseudorandom functions [6, 36].) Let $\mathcal{F}\colon\ \mathcal{K} \times \mathcal{M} \to \mathcal{R}$ be a family of functions from $\mathcal{M}$ to $\mathcal{R}$ indexed by keys $\mathcal{K}$. Let D_{prf} be a distinguisher for $\mathcal{F}$. Consider the following experiments, where $b \in \{0,1\}$ is a bit:

Experiment $\mathbf{Exp}^{\text{prf-b}}_{\mathcal{F}}(D_{\text{prf}})$
  If $b = 1$ then $K \xleftarrow{\$} \mathcal{K}$; $g \leftarrow \mathcal{F}_K$ else $g \xleftarrow{\$}$ Rand$[\mathcal{M}, \mathcal{R}]$
  Run $D^{g(\cdot)}_{\text{prf}}$
    Reply to $g(M)$ queries as follows: $D_{\text{prf}} \Leftarrow g(M)$
  Until D_{prf} returns a bit d
  Return d

We define the PRF-advantage of the adversary D_{prf} as

$$\mathbf{Adv}^{\text{prf}}_{\mathcal{F}}(D_{\text{prf}}) \ = \ \Pr\left[\, \mathbf{Exp}^{\text{prf-1}}_{\mathcal{F}}(D_{\text{prf}}) = 1 \,\right] - \Pr\left[\, \mathbf{Exp}^{\text{prf-0}}_{\mathcal{F}}(D_{\text{prf}}) = 1 \,\right] .$$

In the concrete setting [6], we say that $\mathcal{F}$ is a *secure pseudorandom function* (PRF-secure) if $\mathbf{Adv}^{\text{prf}}_{\mathcal{F}}(D_{\text{prf}})$ is small for all distinguishers D_{prf} using reasonable resources. ∎

2.3 Pseudorandom Permutations (Block Ciphers)

We formalize the notion of pseudorandom permutations (block ciphers) following [6, 56, 63]. Let $\mathcal{F}\colon\ \mathcal{K} \times \mathcal{M} \to \mathcal{R}$ be a family of functions from $\mathcal{M}$ to $\mathcal{R}$ indexed

by keys $\mathcal{K}$. $\mathcal{F}$ is a family of permutations (i.e., a *block cipher*) if $\mathcal{M} = \mathcal{R}$ and $\mathcal{F}_K(\cdot)$ is a permutation on $\mathcal{M}$ for each $K \in \mathcal{K}$. If $\mathcal{F}$ is a family of permutations, we use $F_K^{-1}(\cdot)$ to denote the inverse of $\mathcal{F}_K(\cdot)$ and we use $\mathcal{F}^{-1}(\cdot,\cdot)$ to denote the function that takes (K, M) as input and computes $F_K^{-1}(M)$. Let Perm$[\mathcal{M}]$ denote the set of all permutations on $\mathcal{M}$. If L is a positive integer, we use Perm$[L]$ as shorthand for Perm$[\{0,1\}^L]$. Informally, $\mathcal{F}$ is a *secure pseudorandom permutation* (PRP) under *chosen-plaintext attacks* if it is hard for all distinguishers (adversaries) D_{prp} using reasonable resources to distinguish $\mathcal{F}_K(\cdot)$, with a randomly selected key $K \in \mathcal{K}$, from a randomly selected permutation f from Perm$[\mathcal{M}]$. Additionally, $\mathcal{F}$ is a *secure pseudorandom permutation* under *chosen-ciphertext attacks* (a.k.a. *super-pseudorandom permutation* [56] or *strong-PRP* [63]) if it is hard for all distinguishers (adversaries) D_{sprp} using reasonable resources to distinguish $\mathcal{F}_K(\cdot), \mathcal{F}_K^{-1}(\cdot)$, with a randomly selected key $K \in \mathcal{K}$, from a randomly selected permutation f from Perm$[\mathcal{M}]$ and f's inverse. We make this more formal below.

Definition 2.3.1 (Pseudorandom permutations [6, 56, 63].) Let $\mathcal{F}: \mathcal{K} \times \mathcal{M} \to \mathcal{M}$ be a family of permutations on $\mathcal{M}$ indexed by $\mathcal{K}$. Let D_{prp} and D_{sprp} be distinguishers for $\mathcal{F}$. Consider the following experiments, where $b \in \{0,1\}$ is a bit:

Experiment $\mathbf{Exp}_{\mathcal{F}}^{\text{prp-b}}(D_{\text{prp}})$
  If $b = 1$ then $K \xleftarrow{\$} \{0,1\}^k$; $g \leftarrow \mathcal{F}_K$ else $g \xleftarrow{\$}$ Perm$[\mathcal{M}]$
  Run $D_{\text{prp}}^{g(\cdot)}$
    Reply to $g(M)$ queries as follows: $D_{\text{prp}} \Leftarrow g(M)$
  Until D_{prp} returns a bit d
  Return d

Experiment $\mathbf{Exp}_{\mathcal{F}}^{\text{prp-cca-b}}(D_{\text{sprp}})$
  If $b = 1$ then $K \xleftarrow{\$} \{0,1\}^k$; $g \leftarrow \mathcal{F}_K$ else $g \xleftarrow{\$}$ Perm$[\mathcal{M}]$
  Run $D_{\text{sprp}}^{g(\cdot),g^{-1}(\cdot)}$
    Reply to $g(M)$ queries as follows: $D_{\text{sprp}} \Leftarrow g(M)$
    Reply to $g^{-1}(C)$ queries as follows: $D_{\text{sprp}} \Leftarrow g^{-1}(C)$

Until D_{sprp} returns a bit d

Return d

We respectively define the PRP- and SPRP-advantages of the adversaries D_{prp} and D_{sprp} as

$$\begin{aligned}
\mathbf{Adv}^{\text{prp}}_{\mathcal{F}}(D_{\mathrm{prp}}) &= \Pr\left[\,\mathbf{Exp}^{\text{prp-1}}_{\mathcal{F}}(D_{\mathrm{prp}}) = 1\,\right] - \Pr\left[\,\mathbf{Exp}^{\text{prp-0}}_{\mathcal{F}}(D_{\mathrm{prp}}) = 1\,\right] \\
\mathbf{Adv}^{\text{prp-cca}}_{\mathcal{F}}(D_{\mathrm{sprp}}) &= \Pr\left[\,\mathbf{Exp}^{\text{prp-cca-1}}_{\mathcal{F}}(D_{\mathrm{sprp}}) = 1\,\right] \\
&\quad - \Pr\left[\,\mathbf{Exp}^{\text{prp-cca-0}}_{\mathcal{F}}(D_{\mathrm{sprp}}) = 1\,\right] .
\end{aligned}$$

In the concrete setting [6], we say that $\mathcal{F}$ is a *secure pseudorandom permutation* under *chosen-plaintext attacks* (PRP-secure) if $\mathbf{Adv}^{\text{prp}}_{\mathcal{F}}(D_{\mathrm{prp}})$ is small for all distinguishers D_{prp} using reasonable resources. Similarly, we say that $\mathcal{F}$ is a *secure pseudorandom permutation* under *chosen-ciphertext attacks* (SPRP-secure) if $\mathbf{Adv}^{\text{prp-cca}}_{\mathcal{F}}(D_{\mathrm{sprp}})$ is small for all distinguishers D_{sprp} using reasonable resources. ∎

Most cryptographers believe AES [28] to be an example of a PRP- and SPRP-secure block cipher.

2.4 Symmetric Encryption

We formalize the notion of a symmetric encryption scheme following [4]. A symmetric encryption scheme $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ consists of three algorithms, $\mathcal{K}$, $\mathcal{E}$, and $\mathcal{D}$. The randomized key generation algorithm $\mathcal{K}$ returns a key K from the set $\mathsf{KeySp}_{\mathcal{SE}}$; we write this as $K \xleftarrow{\$} \mathcal{K}$. The encryption algorithm, which could be both randomized or stateful, takes a key $K \in \mathsf{KeySp}_{\mathcal{SE}}$ and a plaintext $M \in \{0,1\}^*$ as input and returns a ciphertext $C \in \{0,1\}^*$ or the error code $\perp$; we write this as $C \xleftarrow{\$} \mathcal{E}_K(M)$. The decryption algorithm, which is stateless and deterministic, takes the key $K \in \mathsf{KeySp}_{\mathcal{SE}}$ and a string $C \in \{0,1\}^*$ as input and returns either the corresponding plaintext M or the error code $\perp$; we write this as $x \leftarrow \mathcal{D}_K(C)$. The consistency requirement is that, regardless

of the state of the encryptor, for all keys $K \in \mathsf{KeySp}_{\mathcal{SE}}$ and messages $M \in \{0,1\}^*$, if $\mathcal{E}_K(M)$ returns C, then either $C = \perp$ or $\mathcal{D}_K(C) = M$.

The formal notions of *privacy* under *chosen-plaintext* and *chosen-ciphertext attacks* for symmetric encryption schemes come from [4]. Intuitively, the notions of privacy measure the ability of an adversary to distinguish between the encryption of two sequences of messages. For the chosen-ciphertext privacy notion, we also give the adversary the ability to decrypt any string of its choice, assuming that the string does not correspond to a ciphertext generated by the encryptor (since otherwise the adversary could trivially learn which sequence of messages was encrypted). The idea is that if an adversary cannot effectively distinguish between the encryption of two different sequences of messages, then it certainly cannot accomplish greater tasks, like decrypting arbitrary ciphertexts or figuring out the encryption key. Toward making this definition more formal, let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ again denote a symmetric encryption scheme. For a key $K \in \mathsf{KeySp}_{\mathcal{SE}}$ and bit $b \in \{0,1\}$, let $\mathcal{E}_K(\mathcal{LR}(\cdot,\cdot,b))$ denote a *left-or-right encryption oracle* (LR encryption oracle) that takes input $M_0, M_1 \in \{0,1\}^*$, where $|M_0| = |M_1|$, and returns $\mathcal{E}_K(M_b)$, the encryption of M_b. In pseudocode, we define $\mathcal{E}_K(\mathcal{LR}(\cdot,\cdot,b))$ as follow:

Oracle $\mathcal{E}_K(\mathcal{LR}(M_0, M_1, b))$ // $|M_0| = |M_1|$
  $C \xleftarrow{\$} \mathcal{E}_K(M_b)$
  Return C

The definitions of privacy follow.

Definition 2.4.1 (Privacy for symmetric encryption schemes [4].) Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme. Let A_{cpa} be an adversary that has access to a left-or-right encryption oracle $\mathcal{E}_K(\mathcal{LR}(\cdot,\cdot,b))$; let A_{cca} be an adversary that has access to a left-or-right encryption oracle and a decryption oracle $\mathcal{D}_K(\cdot)$. Each adversary returns a bit. Consider the experiments below, where $b \in \{0,1\}$ is a bit:

Experiment $\mathbf{Exp}^{\text{priv-cpa-b}}_{\mathcal{SE}}(A_{\text{cpa}})$
  $K \xleftarrow{\$} \mathcal{K}$

Run $A_{\text{cpa}}^{\mathcal{E}_K(\mathcal{LR}(\cdot,\cdot,b))}$

Reply to $\mathcal{E}_K(\mathcal{LR}(M_0, M_1, b))$ queries as follows:

$C \xleftarrow{\$} \mathcal{E}_K(M_b)\ ;\ A_{\text{cpa}} \Leftarrow C$

Until A_{cpa} returns a bit d

Return d

Experiment $\mathbf{Exp}_{\mathcal{SE}}^{\text{priv-cca-b}}(A_{\text{cca}})$

$K \xleftarrow{\$} \mathcal{K}$

Run $A_{\text{cca}}^{\mathcal{E}_K(\mathcal{LR}(\cdot,\cdot,b)),\mathcal{D}_K(\cdot)}$

Reply to $\mathcal{E}_K(\mathcal{LR}(M_0, M_1, b))$ queries as follows:

$C \xleftarrow{\$} \mathcal{E}_K(M_b)\ ;\ A_{\text{cca}} \Leftarrow C$

Reply to $\mathcal{D}_K(C)$ queries as follows: $M \leftarrow \mathcal{D}_K(C)\ ;\ A_{\text{cca}} \Leftarrow M$

Until A_{cca} returns a bit d

Return d

We require that, for all queries (M_0, M_1) to $\mathcal{E}_K(\mathcal{LR}(\cdot,\cdot,b))$, $|M_0| = |M_1|$. For the PRIV-CCA experiment, $\mathbf{Exp}_{\mathcal{SE}}^{\text{priv-cca-b}}(A_{\text{cca}})$, we require that A_{cca} not query $\mathcal{D}_K(\cdot)$ on a ciphertext previously returned by $\mathcal{E}_K(\mathcal{LR}(\cdot,\cdot,b))$. We respectively define the PRIV-CPA- and PRIV-CCA-advantages of the adversaries as

$$\begin{aligned}
\mathbf{Adv}_{\mathcal{SE}}^{\text{priv-cpa}}(A_{\text{cpa}}) &= \Pr\left[\mathbf{Exp}_{\mathcal{SE}}^{\text{priv-cpa-1}}(A_{\text{cpa}}) = 1\right] - \Pr\left[\mathbf{Exp}_{\mathcal{SE}}^{\text{priv-cpa-0}}(A_{\text{cpa}}) = 1\right] \\
\mathbf{Adv}_{\mathcal{SE}}^{\text{priv-cca}}(A_{\text{cca}}) &= \Pr\left[\mathbf{Exp}_{\mathcal{SE}}^{\text{priv-cca-1}}(A_{\text{cca}}) = 1\right] - \Pr\left[\mathbf{Exp}_{\mathcal{SE}}^{\text{priv-cca-0}}(A_{\text{cca}}) = 1\right] .
\end{aligned}$$

In the concrete setting [6], we say that $\mathcal{SE}$ is *privacy-preserving* (*indistinguishable*) under *chosen-plaintext attacks* (PRIV-CPA-secure) if $\mathbf{Adv}_{\mathcal{SE}}^{\text{priv-cpa}}(A_{\text{cpa}})$ is small for all adversaries A_{cpa} using reasonable resources. Similarly, we say that $\mathcal{SE}$ is *privacy-preserving* under *chosen-ciphertext attacks* (PRIV-CCA-secure) if $\mathbf{Adv}_{\mathcal{SE}}^{\text{priv-cca}}(A_{\text{cca}})$ is small for all adversaries A_{cca} using reasonable resources. ∎

To provide more intuition behind these definitions, consider the case of indistinguishability under chosen-plaintext attacks. In words, we define the chosen-plaintext

privacy advantage of the adversary A_{cpa} with access to $\mathcal{E}_K(\mathcal{LR}(\cdot,\cdot,b))$, where K is selected at random, as the probability that A_{cpa} guesses that b is 1 when b is actually 1 minus the probability that A_{cpa} guesses that b is 1 when b is actually 0. Intuitively, $\mathbf{Adv}^{\text{priv-cpa}}_{\mathcal{SE}}(A_{\text{cpa}})$ and $\mathbf{Adv}^{\text{priv-cca}}_{\mathcal{SE}}(A_{\text{cca}})$ are small (close to 0) if an adversary has a hard time in guessing the bit b, but $\mathbf{Adv}^{\text{priv-cpa}}_{\mathcal{SE}}(A_{\text{cpa}})$ and $\mathbf{Adv}^{\text{priv-cca}}_{\mathcal{SE}}(A_{\text{cca}})$ are large (close to 1) if an adversary has an easy time in guessing the bit b.

CTR mode and CBC mode [4, 30] are examples of two popular block cipher-based symmetric encryption schemes that are provably PRIV-CPA-secure assuming that the underlying block cipher is PRP-secure.

2.5 Message Authentication

We formalize the notion of a message authentication scheme following [6, 10]. A *message authentication scheme* (MAC) $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ consists of three algorithms. The randomized key generation algorithm returns a key K from the set $\mathsf{KeySp}_{\mathcal{MA}}$; we write this as $K \xleftarrow{\$} \mathcal{K}$. The tagging algorithm, which may be both randomized and stateful, takes a key $K \in \mathsf{KeySp}_{\mathcal{MA}}$ and a message $M \in \{0,1\}^*$ and returns a tag $\tau \in \{0,1\}^*$; we write this as $\tau \xleftarrow{\$} \mathcal{T}_K(M)$. The deterministic and stateless verification algorithm takes a key $K \in \mathsf{KeySp}_{\mathcal{MA}}$, a message $M \in \{0,1\}^*$, and a candidate tag $\tau \in \{0,1\}^*$ and returns a bit b; we write $b \leftarrow \mathcal{V}_K(M,\tau)$. For any key $K \in \mathsf{KeySp}_{\mathcal{MA}}$ and message $M \in \{0,1\}^*$, and for any internal state of $\mathcal{T}_K$, we require that $\mathcal{V}_K(M, \mathcal{T}_K(M)) = 1$.

We consider a secure MAC $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ to be one that is *strongly unforgeable under chosen-message attacks* [10]. We consider a game in which a forger F is given access to a tagging oracle $\mathcal{T}_K(\cdot)$ and a verification oracle $\mathcal{V}_K(\cdot)$. The forger is allowed arbitrary queries to the oracles and wins if it can find a pair (M,τ) such that $\mathcal{V}_K(M,\tau) = 1$ but τ was never returned by $\mathcal{T}_K(\cdot)$ as a tag for M. We denote the advantage of this forger as $\mathbf{Adv}^{\text{uf}}_{\mathcal{MA}}(F)$. Although this notion is in general stronger than the standard notion of unforgeability [6], we note that any pseudorandom function is a

strongly unforgeable MAC, and most practical MACs seem to be strongly unforgeable. A more formal presentation of the definition follows.

Definition 2.5.1 (Strong unforgeability of message authentication schemes [10].) Let $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ be a message authentication scheme. Let F be a forger with access to a tagging oracle $\mathcal{T}_K(\cdot)$ and a verification oracle $\mathcal{V}_K(\cdot,\cdot)$. Consider the following experiment:

Experiment $\mathbf{Exp}^{\text{uf}}_{\mathcal{MA}}(F)$
  $K \xleftarrow{\$} \mathcal{K}$; $S \leftarrow \emptyset$
  Run $F^{\mathcal{T}_K(\cdot),\mathcal{V}_K(\cdot,\cdot)}$
    Reply to $\mathcal{T}_K(M)$ queries as follows:
      $\tau \xleftarrow{\$} \mathcal{T}_K(M)$; $S \leftarrow S \cup \{(M,\tau)\}$; $F \Leftarrow \tau$
    Reply to $\mathcal{V}_K(M,\tau)$ queries as follows:
      $v \leftarrow \mathcal{V}_K(M,\tau)$
      If $v = 1$ and $(M,\tau) \notin S$ then return 1
      $F \Leftarrow v$
  Until F halts
  Return 0

We define the UF-advantage of F in *forging* a message-tag pair as

$$\mathbf{Adv}^{\text{uf}}_{\mathcal{MA}}(F) = \Pr\left[\, \mathbf{Exp}^{\text{uf}}_{\mathcal{MA}}(F) = 1 \,\right] .$$

In the concrete setting [6], we say that $\mathcal{MA}$ is a *strongly unforgeable* (UF-secure) if $\mathbf{Adv}^{\text{uf}}_{\mathcal{MA}}(F)$ is small for all forgers F using reasonable resources. ∎

If the message authentication scheme $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ has a stateless and deterministic tagging function $\mathcal{T}: \ \mathsf{KeySp}_{\mathcal{MA}} \times \{0,1\}^* \rightarrow \mathcal{R}$ for some range $\mathcal{R}$, then we can apply the definition of a pseudorandom function from Section 2.2 to $\mathcal{MA}$; for $\mathbf{Exp}^{\text{prf-1}}_{\mathcal{MA}}(D_{\text{prf}})$ we select K via $\mathcal{K}$. Moreover, if a MAC is PRF-secure, then it is also UF-secure [6]. Popular provably pseudorandom (and therefore strongly unforgeable)

message authentication schemes include OMAC [41] and HMAC [3, 53, 2], the former secure assuming that the underlying block cipher is PRP-secure [41] and the latter secure assuming that the underlying compression is a secure pseudorandom function under a small class of related-key attacks [2, 7, 19].

2.6 Authenticated Encryption

The notion of a symmetric *authenticated encryption* scheme was first formally introduced by Katz and Yung [47], Bellare and Rogaway [11], and Bellare and Namprempre [10]. An authenticated encryption scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is like a traditional symmetric encryption scheme (Section 2.4), with the same syntax and consistency requirement, but with the goal of providing *both* authenticity (integrity) and privacy. The definitions of privacy for authenticated encryption schemes are also the same as the definitions of privacy for symmetric encryption schemes from Section 2.4. The formal notions of integrity for authenticated encryption schemes in [10, 11, 47] are based on the notion of unforgeability for message authentication schemes from [6]. Intuitively, one notion of integrity, AUTHC, measures an adversary's inability to trick the decryption algorithm into accepting a ciphertext that the encryption algorithm did not generate. The AUTHC notion is also called *integrity of ciphertexts*. A weaker notion, AUTHP or *integrity of plaintexts*, measures an adversary's inability to trick the decryption algorithm into accepting a ciphertext that decrypts to a message that the encryptor did not encrypt. These integrity definitions are described more formally below.

Definition 2.6.1 (Integrity for authenticated encryption schemes [10, 11, 47].) Let $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an AE scheme. Let $F_{\mathrm{p}}, F_{\mathrm{c}}$ be forgers with access to an encryption oracle $\mathcal{E}_K(\cdot)$ and a decryption-verification oracle $\mathcal{D}^*_K(\cdot)$; the latter, on input C, invokes $\mathcal{D}_K(C)$ and returns 1 (i.e., accepts) if $\mathcal{D}_K(C) \neq \perp$ and 0 (i.e., rejects) otherwise. Consider the experiments:

Experiment $\mathbf{Exp}^{\mathrm{authp}}_{\mathcal{AE}}(F_{\mathrm{p}})$

$K \xleftarrow{\$} \mathcal{K}$; $S \leftarrow \emptyset$

Run $F_{\rm p}^{\mathcal{E}_K(\cdot),\mathcal{D}_K^*(\cdot)}$

Reply to $\mathcal{E}_K(M)$ queries as follows:

$C \xleftarrow{\$} \mathcal{E}_K(M)$; $S \leftarrow S \cup \{M\}$; $F_{\rm p} \Leftarrow C$

Reply to $\mathcal{D}_K^*(C)$ queries as follows:

$M \leftarrow \mathcal{D}_K(C)$

If $M \neq \perp$ and $M \notin S$ then return 1

If $M \neq \perp$ then $F_{\rm p} \Leftarrow 1$ else $F_{\rm p} \Leftarrow 0$

Until $F_{\rm p}$ halts

Return 0

Experiment $\mathbf{Exp}^{\text{authc}}_{\mathcal{AE}}(F_{\rm c})$

$K \xleftarrow{\$} \mathcal{K}$; $S \leftarrow \emptyset$

Run $F_{\rm c}^{\mathcal{E}_K(\cdot),\mathcal{D}_K^*(\cdot)}$

Reply to $\mathcal{E}_K(M)$ queries as follows:

$C \xleftarrow{\$} \mathcal{E}_K(M)$; $S \leftarrow S \cup \{C\}$; $F_{\rm c} \Leftarrow C$

Reply to $\mathcal{D}_K^*(C)$ queries as follows:

$M \leftarrow \mathcal{D}_K(C)$

If $M \neq \perp$ and $C \notin S$ then return 1

If $M \neq \perp$ then $F_{\rm c} \Leftarrow 1$ else $F_{\rm c} \Leftarrow 0$

Until $F_{\rm c}$ halts

Return 0

We respectively define the AUTHP- and AUTHC-advantages of $F_{\rm p}$ and $F_{\rm c}$ as

$$\begin{aligned} \mathbf{Adv}^{\text{authp}}_{\mathcal{AE}}(F_{\rm p}) &= \Pr\left[\,\mathbf{Exp}^{\text{authp}}_{\mathcal{AE}}(F_{\rm p}) = 1\,\right] \\ \mathbf{Adv}^{\text{authc}}_{\mathcal{AE}}(F_{\rm c}) &= \Pr\left[\,\mathbf{Exp}^{\text{authc}}_{\mathcal{AE}}(F_{\rm c}) = 1\,\right] . \end{aligned}$$

In the concrete setting [6], we say that *$\mathcal{AE}$ preserves integrity of plaintexts* (AUTHP-secure) if $\mathbf{Adv}^{\text{authp}}_{\mathcal{AE}}(F_{\rm p})$ is small for all forgers $F_{\rm p}$ using reasonable resources. Similarly, we say that *$\mathcal{AE}$ preserves integrity of ciphertexts* (AUTHC-secure) if $\mathbf{Adv}^{\text{authc}}_{\mathcal{AE}}(F_{\rm c})$ is small for all forgers $F_{\rm c}$ using reasonable resources. ∎

2.6.1 Relations Between Notions

Katz and Yung [47] and Bellare and Namprempre [10] prove that if an authenticated encryption scheme preserves privacy under chosen-plaintext attacks (PRIV-CPA-secure) and also preserves integrity of ciphertexts (AUTHC-secure), then it also preserves privacy under chosen-ciphertext attacks (PRIV-CCA-secure). This important result means that it is sufficient for designers of authenticated encryption schemes to focus solely on the PRIV-CPA and AUTHC security properties, even though PRIV-CCA is the principle privacy goal.

The formal statement of this result is below. To briefly interpret the following theorem, the theorem shows that the advantage of an adversary attacking the chosen-ciphertext privacy of $\mathcal{AE}$ is upper-bounded by the advantages of adversaries B and I, using similar resources, in respectively breaking the chosen-plaintext privacy or breaking the authenticity of $\mathcal{AE}$. If we assume that $\mathcal{AE}$ preserves privacy under chosen plaintext attack, then $\mathbf{Adv}^{\text{priv-cpa}}_{\mathcal{AE}}(B)$ must necessarily be small (by definition). Similarly, if we assume that $\mathcal{AE}$ preserves integrity of ciphertexts, then $\mathbf{Adv}^{\text{authc}}_{\mathcal{AE}}(I)$ must also be small. Consequently, if we assume that $\mathcal{AE}$ preserves privacy under chosen-plaintext attacks and also preserves integrity of ciphertexts, then it must preserve privacy under chosen-ciphertext attacks.

Theorem 2.6.2 (If an AE scheme is PRIV-CPA-secure and AUTHC-secure, then it is also PRIV-CCA-secure [10, 47].) Let $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an authenticated encryption scheme. Given any PRIV-CCA adversary A, we can construct an AUTHC adversary I and a PRIV-CPA adversary B, both of which run A as a subroutine, such that

$$\mathbf{Adv}^{\text{priv-cca}}_{\mathcal{AE}}(A) \leq \mathbf{Adv}^{\text{priv-cpa}}_{\mathcal{AE}}(B) + 2 \cdot \mathbf{Adv}^{\text{authc}}_{\mathcal{AE}}(I)$$

and I and B use the same resources as A. ∎

Bellare and Namprempre [10] prove other relations, e.g., that an AE scheme that preserves privacy under chosen-ciphertext attacks (PRIV-CCA-secure) may not provide

integrity of ciphertexts (not AUTHC-secure). While this particular relationship is important from a foundational perspective, it is less useful when trying to prove the security of an authenticated encryption scheme. Bellare and Namprempre [10] also show that AUTHP in combination with PRIV-CPA, does not imply PRIV-CCA. Because AUTHP and PRIV-CPA do not imply PRIV-CCA, the AUTHP notion appears less frequently in the literature than the AUTHC notion.

2.6.2 Definitional Variations

There are a number of variations to the definitions presented above. Rogaway, Bellare, and Black [72] define a notion of chosen-plaintext privacy that is stronger than the PRIV-CPA notion above, though we stress that the community still believes that the standard PRIV-CPA notion captures an appropriate level of security; the notion in the RBB paper [72] measures an adversary's ability to distinguish between the encryption of real messages from random strings of the same lengths as the real ciphertexts. Rogaway [69] also introduces the notion of an *authenticated encryption with associated data* (AEAD) scheme, which extends the definition of an authenticated encryption scheme to allow for the scheme to authenticate more data than it encrypts; AEAD schemes are desirable when processing network packets with headers that need to be authenticated but not encrypted. Canetti and Krawczyk [25, 26, 52] also model privacy- and authenticity-providing symmetric protocols as part of their universal composability secure channels work. We introduce additional definitions later in this dissertation.

2.6.3 Generic Composition

Although the formal definitions of an authenticated encryption scheme only recently appeared in the cryptographic literature [10, 11, 47], applied cryptographers have been trying to create authenticated encryption schemes for years, and one of the most popular design strategies has been to combine standard chosen-plaintext privacy-only (PRIV-CPA-only) encryption schemes with standard authenticity-only (UF-only) mes-

sage authentication schemes. For example, IPsec, SSL/TLS, and SSH all use this basic approach. Bellare and Namprempre [10] and Krawczyk [52] were the first to formally consider the natural approaches for creating authenticated encryption schemes from standard encryption schemes and standard message authentication schemes as black boxes. We summarize their results here, emphasizing the fact that although one can construct a secure authenticated encryption scheme from a secure encryption scheme and a secure MAC, simply combining a secure encryption scheme and a secure MAC is *not* guaranteed to yield a secure authenticated encryption scheme.

Bellare and Namprempre [10] and Krawczyk [52] identified three paradigms for constructing composition-based authenticated encryption schemes: Encrypt-and-MAC, MAC-then-Encrypt, and Encrypt-then-MAC. These constructions are so-named because of the order in which they run the underlying encryption and message authentication algorithms. These types of constructions are called "generic composition" constructions since they treat the underlying components generically, i.e., as black boxes.

For all of the following, let $\mathcal{SE} = (\mathcal{K}_e, \mathcal{E}, \mathcal{D})$ be an encryption scheme and let $\mathcal{MA} = (\mathcal{K}_m, \mathcal{T}, \mathcal{V})$ be a message authentication scheme. For simplicity, assume that $\mathcal{E}$ never outputs the error code $\perp$ and that all the tags output by $\mathcal{T}$ are the same length, i.e., t-bit strings for some constant t.

Encrypt-and-MAC. Given $\mathcal{SE}$ and $\mathcal{MA}$, the composite Encrypt-and-MAC construction $\mathcal{AE} = (\overline{\mathcal{K}}, \overline{\mathcal{E}}, \overline{\mathcal{D}})$ is defined as follows:

Algorithm $\overline{\mathcal{K}}$	Algorithm $\overline{\mathcal{E}}_{\langle K_e, K_m \rangle}(M)$	Algorithm $\overline{\mathcal{D}}_{\langle K_e, K_m \rangle}(C)$
$K_e \xleftarrow{\$} \mathcal{K}_e$	$C' \xleftarrow{\$} \mathcal{E}_{K_e}(M)$	Parse C as $C' \| \tau$
$K_m \xleftarrow{\$} \mathcal{K}_m$	$\tau \xleftarrow{\$} \mathcal{T}_{K_m}(M)$	$M \leftarrow \mathcal{D}_{K_e}(C')$
Return $\langle K_e, K_m \rangle$	$C \leftarrow C' \| \tau$	$v \leftarrow \mathcal{V}_{K_m}(M, \tau)$
	Return C	If $v = 1$ return M
		Else return $\perp$

MAC-then-Encrypt. Given $\mathcal{SE}$ and $\mathcal{MA}$, the composite MAC-then-Encrypt construction $\mathcal{AE} = (\overline{\mathcal{K}}, \overline{\mathcal{E}}, \overline{\mathcal{D}})$ is defined as follows:

Algorithm $\overline{\mathcal{K}}$	Algorithm $\overline{\mathcal{E}}_{\langle K_e, K_m\rangle}(M)$	Algorithm $\overline{\mathcal{D}}_{\langle K_e, K_m\rangle}(C)$
$K_e \xleftarrow{\$} \mathcal{K}_e$	$\tau \xleftarrow{\$} \mathcal{T}_{K_m}(M)$	$M' \leftarrow \mathcal{D}_{K_e}(C)$
$K_m \xleftarrow{\$} \mathcal{K}_m$	$C \xleftarrow{\$} \mathcal{E}_{K_e}(M\|\tau)$	Parse M' as $M\|\tau$
Return $\langle K_e, K_m\rangle$	Return C	$v \leftarrow \mathcal{V}_{K_m}(M, \tau)$
		If $v = 1$ return M
		Else return $\perp$

Encrypt-then-MAC. Given $\mathcal{SE}$ and $\mathcal{MA}$, the composite Encrypt-then-MAC construction $\mathcal{AE} = (\overline{\mathcal{K}}, \overline{\mathcal{E}}, \overline{\mathcal{D}})$ is defined as follows:

Algorithm $\overline{\mathcal{K}}$	Algorithm $\overline{\mathcal{E}}_{\langle K_e, K_m\rangle}(M)$	Algorithm $\overline{\mathcal{D}}_{\langle K_e, K_m\rangle}(C)$
$K_e \xleftarrow{\$} \mathcal{K}_e$	$C' \xleftarrow{\$} \mathcal{E}_{K_e}(M)$	Parse C as $C'\|\tau'$
$K_m \xleftarrow{\$} \mathcal{K}_m$	$\tau' \xleftarrow{\$} \mathcal{T}_{K_m}(C')$	$M \leftarrow \mathcal{D}_{K_e}(C')$
Return $\langle K_e, K_m\rangle$	$C \leftarrow C'\|\tau'$	$v \leftarrow \mathcal{V}_{K_m}(C', \tau')$
	Return C	If $v = 1$ return M
		Else return $\perp$

The security of the generic composition constructions. Bellare and Namprempre [10] presented the following important results about the above composition paradigms. Krawczyk [52] presented similar results, but under slightly different notions of security. We omit formal theorem statements since they are not necessary for understanding the results.

Encrypt-and-MAC: Even if the underlying encryption and message authentication components are respectively PRIV-CPA- and UF-secure, the composite Encrypt-and-MAC construction may fail to preserve privacy under chosen-plaintext attacks and may fail to provide authenticity. That is, an Encrypt-and-MAC construction built from secure components may fail to be PRIV-CPA- and AUTHC-secure.

Even worse, for most popular MACs, and in particular for any secure stateless and deterministic MAC like CBC-MAC or HMAC, the Encrypt-and-MAC construction composed from that MAC can never be PRIV-CPA-secure, regardless of

the choice of the underlying encryption scheme. The critical problem is that the MAC may not be privacy-preserving and, therefore, the tag τ may leak information about the original message M.

MAC-then-Encrypt: If the underlying encryption scheme is PRIV-CPA-secure, then the resulting MAC-then-Encrypt construction will also be PRIV-CPA-secure. On the other hand, even if the underlying encryption and message authentication components are respectively PRIV-CPA-secure and UF-secure, the composite MAC-then-Encrypt construction may fail to be PRIV-CCA- and AUTHC-secure.

Encrypt-then-MAC: If the underlying encryption scheme is PRIV-CPA-secure and if the underlying message authentication scheme is UF-secure, then the resulting Encrypt-then-MAC construction will be PRIV-CPA-secure and AUTHC-secure, and thus also PRIV-CCA-secure by the relation discussed in Section 2.6.1.

Because of these results, Bellare and Namprempre and Krawczyk advise that future composition-based authenticated encryption schemes should use the Encrypt-then-MAC method instead of the Encrypt-and-MAC and MAC-then-Encrypt methods. In Chapter 3, however, we show that it is possible to build a secure authenticated encryption scheme based on the Encrypt-and-MAC paradigm; the trick is to deviate slightly from the exact Encrypt-and-MAC construction shown above.

2.6.4 Encryption with Redundancy

In addition to the generic composition approach for creating authenticated encryption schemes, another popular approach in practice is to combine a standard privacy-only encryption scheme with an unkeyed redundancy function. An example unkeyed redundancy functions might be a 32-bit CRC or a cryptographic hash function like SHA-1, and example protocols built according to this approach are the IEEE 802.11 WEP protocol and version 1.5 of the SSH protocol.

The encryption with unkeyed redundancy approach works as follows. Let $\mathcal{SE} = (\mathcal{K}_e, \mathcal{E}, \mathcal{D})$ be an encryption scheme and let $\mathcal{H}\colon\ \{0,1\}^* \rightarrow \{0,1\}^t$ be an unkeyed

redundancy function. Then the resulting encryption with redundancy scheme $\mathcal{AE} = (\overline{\mathcal{K}}, \overline{\mathcal{E}}, \overline{\mathcal{D}})$ composed from $\mathcal{H}$ and $\mathcal{SE}$ is defined as:

Algorithm $\overline{\mathcal{K}}$	Algorithm $\overline{\mathcal{E}}_K(M)$	Algorithm $\overline{\mathcal{D}}_K(C)$
$K \xleftarrow{\$} \mathcal{K}_e$	$h \leftarrow \mathcal{H}(M)$	$M' \leftarrow \mathcal{D}_K(C)$
Return K	$C \xleftarrow{\$} \mathcal{E}_K(M \| h)$	Parse M' as $M \| h$
	Return C	If $\mathcal{H}(M) = h$ return M
		Else return $\perp$

Unfortunately, the attacks in Bellovin [14] against CBC encryption with redundancy and the attacks in Borisov, Goldberg, and Wagner [24] against WEP show that even if $\mathcal{SE}$ is provably PRIV-CPA-secure, the resulting encryption with redundancy construction may fail to protect the authenticity of encapsulated messages, i.e., $\mathcal{AE}$ may fail to be AUTHC-secure. This means that, in practice, the encryption with redundancy approach should be avoided, at least if the redundancy function is unkeyed and if we are only assuming the standard PRIV-CPA property on $\mathcal{SE}$.

An and Bellare [1] ask whether the security of the encryption with redundancy approach changes if we assume different properties of the underlying encryption scheme or the hash function. For example, what if we assume that $\mathcal{SE}$ is not only PRIV-CPA-secure, but also PRIV-CCA-secure? Or what if the redundancy function is keyed? For the latter, we might also consider what happens if the redundancy function's key is given to the adversary. For the former, recall that a PRIV-CCA-secure encryption scheme may not be AUTHC-secure, which makes the question of whether an encryption with redundancy scheme based on a PRIV-CCA-secure encryption scheme is AUTHC-secure interesting. If the redundancy function is keyed but the adversary gets access to the redundancy function's key, or if the redundancy function is unkeyed, then An and Bellare prove that even assuming PRIV-CCA-security of the underlying encryption scheme is insufficient to guarantee AUTHC-security of the resulting construction. If the redundancy function is keyed and the key is kept secret, then the construction is similar to the MAC-then-Encrypt construction in Section 2.6.3 and, for the same reasons, even if $\mathcal{SE}$ is PRIV-CPA-secure, the resulting construction may fail to be AUTHC-secure.

On the positive side, if $\mathcal{SE}$ is PRIV-CCA-secure, and if the keyed function satisfies a weak notion of unforgeability, then the resulting construction will be both AUTHC- and PRIV-CPA-secure, and therefore a secure authenticated encryption scheme. This positive result, however, is mostly of foundational interest since, in practice, most basic encryption schemes from which we might consider creating authenticated encryption schemes are not PRIV-CCA-secure, but only PRIV-CPA-secure. As another positive result, An and Bellare [1] introduce a specific PRIV-CPA-secure encryption scheme, based on CBC mode encryption, that in combination with certain types of keyed redundancy functions yields a secure authenticated encryption scheme.

2.6.5 Encode-then-Encipher

All of the provably-secure authenticated encryption mechanisms described thus far have encryption algorithms that, on input a message M, return a ciphertext C where the length of C is strictly larger than the length of M. Unfortunately, when we wish to add authenticated encryption to a legacy application, we may not be able to afford the luxury of changing the packet format and increasing its length. Thus rises the question of whether it is possible to achieve authenticated encryption while keeping the lengths of the ciphertexts equal to the lengths of the plaintexts. In general the answer to this question is no since any length-preserving invertible transformation with a stateless inverter must be a permutation, and therefore not privacy-preserving (encrypting the same message twice will always produce the same output).

Bellare and Rogaway [11] step back and look at this problem from a different perspective. Specifically, they ask what happens if the data is already "highly structured," e.g., perhaps the portion of the legacy protocol that we wish to encrypt contains a sequence number, a length field, application data, and a CRC of the preceding three fields. Bellare and Rogaway show that in some cases it *is* possible to achieve authenticated encryption of application data by applying a keyed length-preserving operation (a *cipher*) to the structured strings containing application data. Bellare and Rogaway call their approach to authenticated encryption the Encode-then-Encipher paradigm.

The Encode-then-Encipher building blocks. Bellare and Rogaway model the structured portion of a legacy protocol as an *encoding* of the higher-level application data. Specifically, an *encoding scheme* $\mathcal{EC} = (\mathsf{Encode}, \mathsf{Decode})$ is a pair of *unkeyed* algorithms. The encoding algorithm Encode, which may be stateful or randomized, on input a message $M \in \{0,1\}^*$, returns a string $M' \in \{0,1\}^*$, and we write this as $M' \xleftarrow{\$} \mathsf{Encode}(M)$. We require that for all $M_1, M_2 \in \{0,1\}^*$, if $|M_1| = |M_2|$, then $|\mathsf{Encode}(M_1)| = |\mathsf{Encode}(M_2)|$. The encoding algorithm models the process of taking application data and loading it into a structured packet like the one mentioned above. The decoding algorithm Decode, which is stateless and deterministic, takes as input a string $M' \in \{0,1\}^*$ and returns either a string $M \in \{0,1\}^*$ or the distinguished symbol $\perp$, and we write this as $M \leftarrow \mathsf{Decode}(M')$. We require that for all $M \in \{0,1\}^*$ and for all states of and random tapes for Encode, $\mathsf{Decode}(\mathsf{Encode}(M)) = M$. The decoding algorithm models the process of extracting application data from a structured packet. We discuss the security goals for encoding schemes later.

The other component of an Encode-then-Encipher construction is a cipher, which shares similar properties with block ciphers. Let $\mathcal{F}\colon\ \mathsf{KeySp}_{\mathcal{F}} \times \{0,1\}^* \to \{0,1\}^*$ be a function. The function $\mathcal{F}$ is a *cipher* if for all $K \in \mathsf{KeySp}_{\mathcal{F}}$, $\mathcal{F}_K$ is a length-preserving permutation on $\{0,1\}^*$, and in this case $\mathcal{F}_K^{-1}$ denotes the inverse of $\mathcal{F}_K$. Let LPerm$[\mathcal{M}]$ denote the set of all length-preserving permutations on the set $\mathcal{M} \subseteq \{0,1\}^*$. A cipher $\mathcal{F}$ is pseudorandom under chosen-ciphertext attacks if all adversaries using reasonable resources have a hard time distinguishing between oracle access to $\mathcal{F}_K$ and $\mathcal{F}_K^{-1}$, where K is a randomly selected key, and oracle access to a randomly selected element of LPerm$[\{0,1\}^*]$ and its inverse. More formally, if A is an adversary with access to two oracles and that returns a bit, we define the SPRP-advantage of A in breaking the pseudorandomness under chosen-ciphertext attacks of $\mathcal{F}$ as

$$\begin{aligned}\mathbf{Adv}^{\text{sprp}}_{\mathcal{F}}(A) \;&=\; \Pr\left[\, K \xleftarrow{\$} \mathsf{KeySp}_{\mathcal{F}} \;:\; A^{\mathcal{F}_K(\cdot),\mathcal{F}_K^{-1}(\cdot)} = 1 \,\right] \\ &\quad - \Pr\left[\, \pi \xleftarrow{\$} \text{LPerm}[\{0,1\}^*] \;:\; A^{\pi(\cdot),\pi^{-1}(\cdot)} = 1 \,\right] .\end{aligned}$$

In the concrete setting [6], we say that $\mathcal{F}$ is *pseudorandom under chosen-ciphertext*

attacks or is SPRP-secure if the magnitude of the SPRP-advantage of all adversaries using reasonable resources is small. Note that a *block cipher* like AES [28] is a special case of a *cipher* in the sense that, for each key, the latter takes variable-length messages as input, whereas the former only takes inputs of some fixed length, such as 128-bit strings. Moreover, provably SPRP-secure (variable length) ciphers like EME* [38] are built from block ciphers.

The Encode-then-Encipher paradigm. Having defined what encoding schemes and ciphers are, it now becomes possible to describe Bellare and Rogaway's Encode-then-Encipher paradigm. Let $\mathcal{F}\colon\ \mathsf{KeySp}_{\mathcal{F}} \times \{0,1\}^* \to \{0,1\}^*$ be a cipher. Let $\mathcal{EC} = (\mathsf{Encode}, \mathsf{Decode})$ be an encoding scheme. Then the composite authenticated encryption scheme $\mathcal{AE} = (\overline{\mathcal{K}}, \overline{\mathcal{E}}, \overline{\mathcal{D}})$ composed from $\mathcal{EC}$ and $\mathcal{F}$ is defined as:

Algorithm $\overline{\mathcal{K}}$	Algorithm $\overline{\mathcal{E}}_K(M)$	Algorithm $\overline{\mathcal{D}}_K(C)$
$K \xleftarrow{\$} \mathsf{KeySp}_{\mathcal{F}}$	$M' \xleftarrow{\$} \mathsf{Encode}(M)$	$M' \leftarrow \mathcal{F}_K^{-1}(C)$
Return K	$C \xleftarrow{\$} \mathcal{F}_K(M')$	$M \leftarrow \mathsf{Decode}(M')$
	Return C	Return M

Although $\overline{\mathcal{D}}$ never explicitly returns $\perp$ in the above construction, recall that $\mathsf{Decode}(M')$ may return $\perp$.

Security of the Encode-then-Encipher paradigm. Bellare and Rogaway prove that if $\mathcal{F}$ is SPRP-secure and if $\mathcal{EC}$ has *collision-resistance* and *low density*, which we describe below, then the composite construction built from $\mathcal{F}$ and $\mathcal{EC}$ is a secure authenticated encryption scheme. Since there exist provably SPRP-secure ciphers $\mathcal{F}$, like the recent EME* [38], and since natural protocol constructions have structured encodings with collision-resistance and low density, Bellare and Rogaway's results provide a means to provably add authenticated encryption to legacy protocols that cannot tolerate any additional packet expansion.

It remains to define what *low density* and *collision-resistance* mean. Let $\mathcal{EC} = (\mathsf{Encode}, \mathsf{Decode})$ be an encoding scheme. The scheme $\mathcal{EC}$ is ϵ-colliding if for any

number q and any (even computationally unbounded) adversary A who asks q queries to an Encode oracle, the probability that two of these queries produce the same valid response is at most $\epsilon(q)$. In the concrete setting, we say that $\mathcal{EC}$ is *collision-resistant* if $\epsilon(q)$ is small for all reasonable values of q. Collision-resistance is easily achieved if the encoding algorithm includes a sequence number or large random string in its output. The scheme $\mathcal{EC}$ is δ-dense if for all positive integers n, the probability over a randomly selected string $M' \in \{0,1\}^n$ that $\mathsf{Decode}(M') \neq \perp$ is less than δ (note that here δ is a constant, but above ϵ is a function). In the concrete setting, we say that $\mathcal{EC}$ has *low density* if δ is small. Low density is easily achieved, for example, by including a CRC or length field in the encoding output and having Decode return $\perp$ if one of these fields is incorrect (or, if we restrict the inputs of Encode and the composite construction to only valid parity-adjusted ASCII strings, having Decode return $\perp$ for any string that is not a valid ASCII sequence).

2.6.6 Block Cipher-Based Constructions

All of the above approaches for constructing provably secure authenticated encryption schemes take provably secure components, like encryption schemes, MACs, or ciphers, and combine or apply them in a way that yields a provably secure authenticated encryption scheme. In turn, these provably secure components are often built from one of cryptography's most basic building blocks, the block cipher. Rather than build authenticated encryption schemes from objects that use block ciphers, from a performance perspective it would seem better to build provably secure authenticated encryption schemes directly from block ciphers.

There are two types of provably secure block cipher-based authenticated encryption schemes. The first type of construction makes a single pass through the data, applying approximately one block cipher operation per block of the plaintext message. The second type of construction is closer to the generic composition constructions, making two passes through the data, but making optimizations along the way. From a technical perspective, the difference between the two types may seem rather artificial since most

two-pass constructions can be converted into single-pass constructions through parallelization or interleaving and since, depending on the metric used, the performance of the second class can rival the performance of single-pass constructions. The difference between the two classes of constructions is, however, especially critical in one arena, namely patents. Multiple parties claim patents on the first class of constructions, but no party claims patents on the second class.

Elements of the first class include RPC and RPC$ [47], IACBC and IAPM [44], XCBC and XECB [35], OCB [72], and AEM [70]. Elements of the latter class include CCM [82], EAX [13], CWC [50], and GCM [60]. All of these constructions are secure assuming that the underlying block cipher is a secure pseudorandom permutation under chosen-ciphertext attacks. IAPM, XECB, OCB, AEM, CWC, and GCM are also all data parallelizable, which makes them attractive in high speed hardware where performance is critical. We discuss the design of CWC in Section 5.

3 The Secure Shell Authenticated Encryption Scheme

Bellare and Namprempre [10] and Krawczyk [52] proved that the Encrypt-and-MAC approach for combining a secure, traditional privacy-only (PRIV-CPA-only) encryption scheme with a secure, stateless and deterministic integrity-only (UF-only) message authentication scheme will *never* yield a secure authenticated encryption scheme; recall also Section 2.6.3. Turning to modern cryptographic protocols, we find that the authenticated encryption core of the Secure Shell (SSH) protocol is, however, based on this insecure Encrypt-and-MAC paradigm. Despite Bellare-Namprempre's and Krawczyk's negative result, we are nevertheless able to *prove* that the overall design of the SSH authenticated encryption scheme is *secure* under reasonable assumptions.

This apparent contradiction arises not from any problem with the theoretical results in the Bellare-Namprempre and Krawczyk works, but from the fact that when real protocols like SSH do not *exactly match* the idealized models on which they are based, the theoretical results about these idealized models are no longer applicable. This situation calls for a broader theory for the construction of authenticated encryption schemes from traditional encryption schemes and MACs — a theory that can capture the complexities of real-world authenticated encryption schemes, like the SSH authenticated encryption core. We initiate such a theory in this chapter through our introduction and

An earlier version of the material in this chapter appears in the ACM Transactions on Information and System Security [8], copyright the ACM.

analysis of the *Encode-then-Encrypt-and-MAC* paradigm, and push these generalizations further in Chapter 4.

As an aside, our analysis of the SSH authenticated encryption scheme did uncover one privacy defect. We stress that this defect is not endemic of the overall design of the SSH authenticated encryption scheme, but is instead due to a poor design choice on the part of the protocol designers: the original specification of the SSH protocol [87] employs an *insecure* underlying encryption scheme. We propose fixes to the SSH protocol that work within the constraints of our provable security results and in particular that do not require changing SSH's overall Encrypt-and-MAC-based approach. Our preferred fixes are now defined as an RFC [9] (standard track document) and are implemented in the OpensSSH application.

3.1 Overview

Conceived as a secure alternative to traditional Unix tools like `rsh` and `rcp`, the IETF standardization body's *Secure Shell* (*SSH*) protocol (version 2.0) has become one of the most popular and widely used cryptographic protocols on the Internet. Because of its popularity and because of the insecurity of programs like `rsh`, `rcp`, and `telnet`, a number of institutions now only allow users to remotely access their facilities using SSH. The cryptographic heart of the SSH protocol is its *Binary Packet Protocol* (*BPP*) [87] — the BPP is responsible for the underlying authenticated encryption of all messages sent between two parties involved in an SSH connection.

Although others have discussed specific properties of the SSH BPP, e.g., problems with not using a MAC [79] or problems with SSH's variant of CBC mode [29], to the best of our knowledge no one has performed a rigorous, provable security-based analysis of the entire SSH BPP authenticated encryption mechanism. Our goal was thus to thoroughly analyze the SSH BPP authenticated encryption scheme and, in the event that we found any problems, to present provably-secure fixes to the protocol. Further motivating our analysis is the fact that the SSH BPP is based on the insecure Encrypt-

and-MAC paradigm.

In order for our fixes to be as useful as possible to the Internet community, when developing our fixes we considered both (1) provable security and (2) efficiency. Additionally, since retroactively modifying existing implementations is often very expensive, we required that our suggested modifications (3) not significantly alter the current SSH specification. For the last point, we note that the creators of SSH had the foresight to design the SSH BPP in a modular way: in particular, it is relatively "easy" to change the SSH BPP's underlying encryption and message authentication modules.

Analysis and provably secure recommendations. The SSH BPP specification states that SSH implementations should use CBC mode encryption [30] with chained initialization vectors (IVs); i.e., the IV used when encrypting a message should be the last block of the previous ciphertext. Unfortunately, CBC mode encryption with chained IVs is not PRIV-CPA-secure [67], and this insecurity extends to SSH; this extension was also reported by Dai [29].

Since CBC mode encryption with chained IVs is not PRIV-CPA-secure, but CBC mode with random IVs is PRIV-CPA-secure [4], a natural fix to the SSH protocol might be to replace the use of chained-IV CBC mode with randomized CBC mode. Unfortunately, we show that doing so is not sufficient. In particular, since the SSH specification does not require the padding to be random, the resulting SSH implementation may be vulnerable to a rather serious reaction-attack, i.e., a privacy attack that works by modifying a sender's ciphertexts and observing the receiver's response.

We next give several secure fixes to the SSH authenticated encryption mechanism. For example, we suggest using randomized CBC mode encryption; the difference between this suggestion and the suggestion in the above paragraph is that we require at least one full block of random padding (this could, however, result in having to encipher more blocks than the previous SSH alternative). We also suggest another CBC variant that does not require additional random padding: CBC mode where the IV is generated by enciphering a counter with a different key. As an additional alternative,

we suggest replacing the underlying encryption scheme with a variant of counter (CTR) mode [32, 55] in which both the sender and receiver maintain a copy of the counter. We also present a framework within which to analyze other possible replacements.

One important advantage of these fixes over the current SSH specification is provable security. Making reasonable assumptions, e.g., that SSH's underlying block cipher is PRP-secure, we show that our alternatives will preserve privacy against adaptive chosen-plaintext and adaptive chosen-ciphertext attacks. We also show that our alternatives will resist forgery, replay, and out-of-order delivery attacks. Finally, we argue that our alternatives, and especially the latter two, also satisfy the other two requirements listed above, namely efficiency and ease of modification.

Theoretical contributions. The previous notions of privacy (PRIV-CPA and PRIV-CCA; Section 2.4 and [4]) and integrity (AUTHP and AUTHC; Section 2.6 and [10, 11, 47]) for authenticated encryption only address encryption schemes with stateless decryption algorithms. The SSH BPP decryption algorithm is, however, stateful. Motivated by a desire to analyze the SSH BPP authenticated encryption scheme, and by the desire to capture the potential "power" of stateful decryption algorithms, we extend the previous notions of privacy and integrity to encryption schemes with stateful decryption algorithms. The aforementioned "power" refers to the fact that if a scheme meets our new notions of security, then, in addition to satisfying the existing notions of privacy and integrity, the scheme will be secure against replay attacks and out-of-order delivery attacks — attacks not captured under the previous models.

One alternative approach to our analysis would have been to model the SSH BPP as a "secure channel," as defined in [25] and characterized in [62], since the notion of secure channels can be applied to encryption schemes with stateful decryption algorithms. We point out that the combination of our notions is stronger than the notion of secure channels: combining a secure key agreement protocol with an authenticated encryption scheme that meets both of our notions will yield a secure channel. Consequently, since our fixes to the SSH BPP provably meet our strong notions, the resulting SSH BPP is

also a secure channel.

We acknowledge that one potential disadvantage of our new notions of security is that they may be "too strong" and that some applications may not require the strength associated with our notions; see [25, 52] for reasons. For those applications, the notion of a secure channel might be more appropriate, as might one of the other notions that we introduce in Chapter 4. Our notions are, however, more appropriate for applications like SSH that do require a higher level of protection such as protection against out-of-order delivery attacks. Finally, we note that side-channel attacks such as those exploiting information leaked through the length of packets or the interval of time between packets (e.g., [27, 76]) are not captured by our security models nor any other provable security models that we are aware of.

Outline. After describing the SSH Binary Packet Protocol in Section 3.2, we present a simple attack against the current SSH specification in Section 3.3. In Section 3.4, we show that "fixing" the SSH BPP in the natural way may result in an insecure protocol. Motivated by the lessons we learned from Sections 3.3 and 3.4, we then present provably-secure fixes to the SSH Binary Packet Protocol in Section 3.5. In Sections 3.6–3.8 we present our provable security results. Finally, in Section 3.9, we discuss our results and make recommendations to the SSH and applied cryptographic communities. We discuss the significance of our earlier attacks and the advantages and disadvantages of switching to our proposed modifications. We also discuss the possibility of changing the SSH BPP from an Encrypt-and-MAC-based construction to an Encrypt-then-MAC-based construction and the possibility of modifying SSH to use a dedicated authenticated encryption scheme such as XCBC [35] or OCB [72].

3.2 The SSH Binary Packet Protocol (SSH BPP)

The SSH Binary Packet Protocol [87] is responsible for encrypting and authenticating all messages between two parties involved in an SSH session. Before beginning

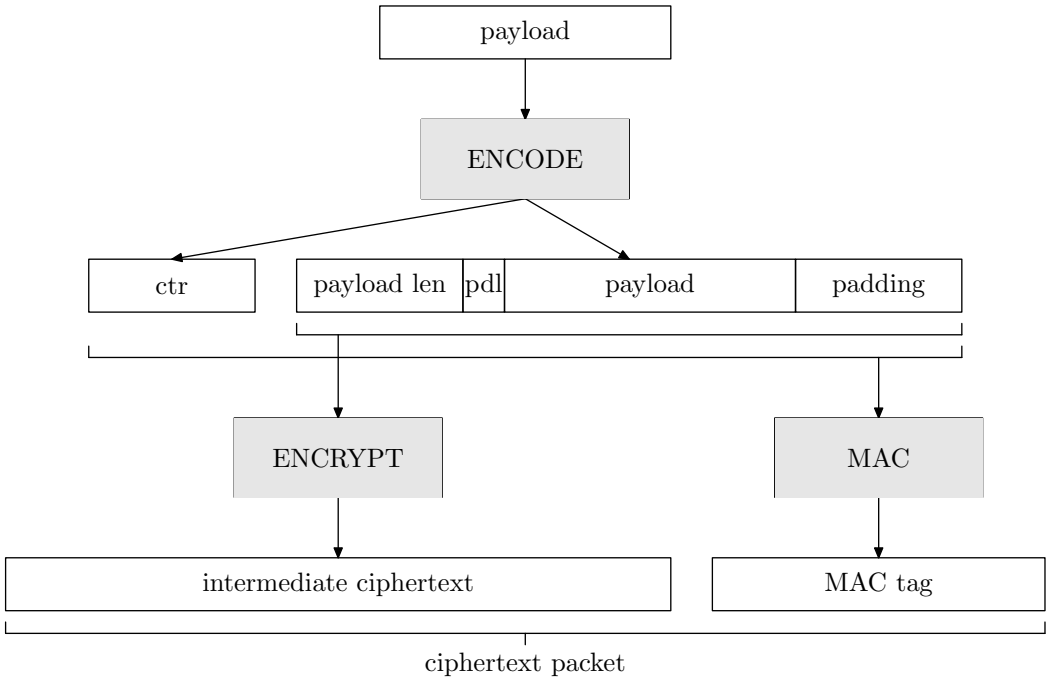

Figure 3.1 The SSH authenticated encryption scheme.

the authenticated encryption portion of an SSH session, a client and a server first agree upon a set of shared symmetric keys (a different set for each direction of a connection). The client and the server also agree upon which encryption and message authentication schemes they wish to use. All of the encryption schemes recommended by the SSH specification [87] are based on CBC mode encryption [30], and all of the recommended message authentication schemes are based on HMAC [53].

Figure 3.1 shows how the SSH authenticated encryption scheme works at a high level. Given a *payload* message (in octets), the SSH BPP encodes that message into an encoded packet consisting of the following fields: a four-octet packet length field containing the length of the remaining encoded packet (in octets), a one-octet padding length field, the payload message, and (possibly random) padding. The length of the total packet must be a multiple of the underlying block cipher's block length, and the padding must be at least four octets long. Although the SSH specification allows up to 255 octets of padding per encoded packet, both implementations that we evalu-

ated, `openssh-2.9p2` and SSH Communications' `ssh-3.0.1`, use the minimum padding necessary. The resulting ciphertext is the concatenation of the encryption of the above encoded packet and the MAC of the above encoded packet prepended with a 32-bit counter. In the following discussions, we try to make clear whether we are referring to the *intermediate ciphertext* output by the underlying encryption scheme or the *ciphertext packet* (the concatenation of the intermediate ciphertext and the MAC tag) output by the SSH BPP.

Decryption is defined in a natural way: the receiver first decrypts the intermediate ciphertext portion of a ciphertext to get an encoded packet. The receiver then prepends a 32-bit counter, which it also maintains, to the encoded packet and determines whether the received MAC tag is valid. If so, the decryptor removes the payload from the encoded packet and delivers the payload to the user (or a higher-level protocol). If the MAC verification fails, the connection is terminated.

The SSH specification recommends the use of CBC mode with inter-packet chaining. This means that, when encrypting an encoded payload, the sender uses as the initialization vector (IV) either the last block of the immediately preceding ciphertext or, when encrypting the first message, an IV computed during the SSH key agreement protocol. We refer to the current instantiation of the SSH protocol as SSH-IPC, or SSH with inter-packet chaining.

3.3 Attacking the Standard Implementation of SSH

There is a simple chosen-plaintext privacy attack against SSH-IPC; this attack was also reported by Dai [29]. The problem with SSH-IPC is that an attacker will know the IV for the next message to be encrypted before the next message is actually encrypted. This means that if an attacker can control the entire first block of the input into SSH-IPC's underlying CBC encryption scheme, it will be able to control the corresponding input to the underlying block cipher. Since a block cipher is deterministic, an attacker could use this to glean information about a previously encrypted message (by

looking to see if some value was ever the input to a previous block cipher invocation).

We describe the attack in slightly more detail. We assume for now that an adversary can control the entire first block of an encoded packet. Suppose that an adversary has a guess G of the first encoded block of the ith packet, and let C_1 be the last CBC block of the $i-1$st intermediate ciphertext. Since we are considering SSH-IPC, the block C_1 was used as the IV when encrypting the ith packet. Let C_2 be the first block of the ith ciphertext. And let C_3 be the last CBC block of the underlying ciphertext the user just output (i.e., the user will use C_3 as its next IV). If the adversary is able to force the user to encrypt the block $C_1 \oplus C_3 \oplus G$, where $\oplus$ is the XOR operation, and if the resulting block is C_2, the adversary knows its guess of for G was correct; otherwise the adversary knows its guess was incorrect.

A small complication arises when mounting this attack against SSH-IPC because the attacker cannot control the entire first block of an encoded message (because the first 40 bits of an encoded packet contain metadata). This means that an attacker may not be able to force a user's underlying CBC scheme to encrypt the block $C_1 \oplus C_3 \oplus G$. An attacker will, however, be able to mount this attack if C_1 and C_3 are identical in the bits that the attacker cannot control. Let l be the block length (in bits) of the underlying block cipher. Since an attacker can control approximately $\lg(l/8)$ bits of the padding length field and approximately $15 - \lg(l/8)$ bits of the packet length field of an encoded message (SSH implementations are only required to support packets with payloads containing less than 2^{15} octets and all packets must be padded to a multiple of the block length), an attacker could mount a variant of the above attack by waiting for a collision on approximately 25 bits (but the adversary's last encryption request may be up to 2^{15} octets long).

3.4 Attacking a Natural "Fix"

The problem with SSH-IPC in Section 3.3 stems from the fact that its underlying encryption scheme is itself vulnerable to chosen-plaintext attacks, i.e., is not PRIV-CPA-

secure. A logical attempt to fix the protocol might therefore be to replace the underlying encryption scheme with randomized CBC mode, i.e., CBC mode in which a new random IV is chosen for each message; this new IV must also be sent with the ciphertext. Randomized CBC mode is provably PRIV-CPA-secure assuming reasonable properties of the underlying block cipher [4]. We refer to an SSH implementation that uses randomized CBC mode as SSH-NPC, or SSH with no packet chaining.

One can prove that SSH-NPC preserves privacy against chosen-plaintext attacks and integrity of plaintexts assuming that a user does not use SSH-NPC to encrypt more than 2^{32} messages with any given key. This proof holds even if the paddings used in encoded packets are not random, a situation allowed by the SSH specification. As the following attack shows, however, even though SSH-NPC with non-random padding preserves privacy against chosen-plaintexts attacks, it does not preserve privacy against chosen-ciphertext attacks.

Reaction attack against SSH-NPC. The SSH specification encourages, although does not require, implementations to use random padding. Unfortunately, when the padding value is fixed, e.g., all zeros, SSH-NPC is susceptible to an easily-mountable reaction attack. Furthermore, one can extend this attack to the case where the padding values are not fixed but short and not hard to predict: an attacker can simply wait until the predicted padding values collide and then use the predicted value to successfully mount an attack. The attack we describe here is similar in spirit to Wagner's attack in [14] and to the attacks in [52, 79]. The term "reaction attack" comes from [39].

The attack proceeds roughly as follows: an attacker intercepts and prevents the delivery of two ciphertexts sent by one party involved in an SSH connection. The adversary then makes a guess about the relationship between the two plaintexts corresponding to the two intercepted ciphertexts. The adversary then uses that guess and those two ciphertexts to create a new "ciphertext," which the adversary then sends to the other party involved in the SSH session. Recall that if the second party does not accept the doctored ciphertext, the connection will be terminated. Thus, by observing the second party's

reaction, an adversary will learn whether its guess was correct. Intuitively, this attack succeeds because an attacker can modify the ciphertext in such a way that if its guess was correct, the ciphertext that the second party receives will verify. If its guess was incorrect, with high probability the ciphertext will not verify.

We now describe the attack in more detail. As before, let $\oplus$ denote the XOR operation, let $\|$ denote the concatenation of two strings, and let l denote the block length (in bits) of the block cipher that SSH-NPC uses in CBC mode. Suppose a user uses SSH-NPC to encrypt two equal-length messages M_1 and M_2 with lengths at most $l-40$ (or messages that are identical after their $l-40$-th bit). For simplicity of exposition, let us assume that the two messages are exactly $l-40$ bits long. Let P_{11} and P_{12} be the first and the second block of the encoded packet corresponding to the payload M_1, respectively. Similarly, let P_{21} and P_{22} be the first and the second block of the encoded packets corresponding to M_2, respectively. The blocks P_{11} and P_{21} correspond to the packet length, the padding length, and the payload fields of the two encoded packets, and the blocks P_{12} and P_{22} correspond to the padding fields. Since we are assuming fixed padding (such as padding with all zeros), the padding blocks P_{12} and P_{22} will be equal.

When SSH-NPC's underlying CBC mode encryption scheme encrypts the first encoded packet $P_{11}\|P_{12}$, it will generate a ciphertext $\sigma_1 = C_{10}\|C_{11}\|C_{12}$. Additionally, SSH-NPC's underlying MAC will generate a tag τ_1 (the MAC being computed over the concatenation of a counter and $P_{11}\|P_{12}$). Similarly, SSH-NPC will generate the CBC ciphertext $C_{20}\|C_{21}\|C_{22}$ and the MAC tag τ_2 for the encoded packet $P_{21}\|P_{22}$. The two blocks C_{10} and C_{20} correspond to the underlying CBC mode's random initialization vectors.

Now assume that the receiver has not yet received the two ciphertexts corresponding to M_1 and M_2. In particular, this means that the recipient's counter is identical to the counter that the sender used when she encrypted the first message. Suppose that the attacker knows either M_1 or M_2 and wants to verify a guess of the other or that the attacker wants to verify a guess of the relationship between M_1 and M_2. Let X be

the value $P_{11} \oplus P_{21} \oplus C_{20}$. The attacker then asks the receiver to decrypt the message $X\|C_{21}\|C_{22}\|\tau_1$. Now recall that the blocks P_{11} and P_{21} both begin with the same 40 bits of header information and that they respectively end in M_1 and M_2. Thus, if the attacker's guess is correct, then $X\|C_{21}\|C_{22}$ will decrypt, via SSH-NPC's underlying CBC scheme, to $P_{11}\|P_{12}$, the MAC tag τ_1 will verify, and the decryptor will accept the message. However, if the attacker's guess is incorrect, $X\|C_{21}\|C_{22}$ will not decrypt to $P_{11}\|P_{12}$, the tag τ_1 will not verify (unless the attacker also succeeds in breaking the security of the underlying MAC scheme), and the SSH-NPC connection will terminate. The adversary, by watching the recipients reaction, therefore learns information about the plaintexts the sender is encrypting.

There are two aspects of this attack that make it easy to mount. First, this attack only requires modifying encrypted packets; no chosen-plaintexts are required. Second, an attacker can learn whether its guess is correct simply by watching the recipient's response. These observations mean that all an attacker needs to perform this attack is the ability to monitor, prevent the delivery of, and inject messages in the encrypted communications between two parties. Similar to Wagner's attack in [14], an adversary can use this attack to, for example, infer the characters that a user types over an interactive SSH-NPC session. Of course, once the attacker makes an incorrect guess, SSH-NPC terminates the connection. Nonetheless, an attacker might still be able to repeat its attack after the user begins a new session.

Information leakage, replay, and out-of-order delivery attacks. Although the SSH draft suggests that an SSH session rekey after every gigabyte of transmitted data, doing so is not required. We caution that if an SSH-NPC (or SSH-IPC) session is not rekeyed frequently enough, then the session will be vulnerable to a number of other attacks. Recall that the SSH binary packet protocol includes a 32-bit counter in each message to be MACed. These attacks make use of the fact that if the SSH connection is not rekeyed frequently enough, then the counter will begin to repeat.

The simple observation exploited by the information leakage attack is the follow-

ing. Recall that SSH generates each MAC using the encoded payload prepended with a counter as an input and then appends the MAC to the intermediate ciphertext to generate a ciphertext packet. As a result, if the underlying MAC algorithm is stateless and deterministic (which many MACs are), then allowing the counter to repeat will leak information about a user's plaintexts (through the MAC). We present the attacks in more details for completeness. Suppose that the underlying message authentication scheme is stateless and deterministic and that the padding is some fixed value. Suppose that an attacker A sees a ciphertext with a MAC tag τ and suspects that the underlying payload is M. To verify its guess, A waits for the sender to encrypt $2^{32} - 1$ more packets and then requests the sender to encrypt the payload M. Let τ' be the MAC tag returned in response to the request. If A's guess is correct, then τ' will equal τ. Otherwise $\tau' \neq \tau$ with very high probability. The attack can also be used to break the privacy of SSH-NPC when SSH-NPC uses random padding. In particular, if the first 2^{32} messages that a user tags result in encoded packets that use the minimum 4 octets of random padding, then an attacker capable of forcing a user to tag an additional 2^{32} chosen-plaintexts will be able to learn information about the user's initial 2^{32} messages. The property used in this attack, namely that tagging with a deterministic MAC leaks information about plaintexts, was also exploited by Bellare and Namprempre [10] and Krawczyk [52] when showing the generic insecurity of all Encrypt-and-MAC constructions using stateless and deterministic MACs; recall also Section 2.6.3.

If the counter is allowed to repeat, SSH-NPC also becomes vulnerable to replay attacks and out-of-order delivery attacks. For replay attacks, once the receiver has decrypted 2^{32} messages, an attacker will be able to convince the receiver to re-accept a previously received message. For out-of-order delivery attacks, after the sender has encrypted more that 2^{32} messages, an attacker will be able to modify the order in which the messages are decrypted.

3.5 Secure Fixes to SSH

We now briefly describe our new SSH instantiations. We show in Section 3.8 that these new alternatives provably meet our strongest notions of security. That is, assuming that these fixes are not used to encrypt more than 2^{32} packets between rekeying, these new constructions will resist chosen-plaintext and chosen-ciphertext privacy attacks as well as forgery, replay, and out-of-order delivery attacks. Security above 2^{32} is not guaranteed because, after 2^{32} packets are encrypted, the SSH BPP's 32-bit internal counter will begin to wrap. We will compare these instantiations of SSH to others and discuss additional possible modifications, including extending the length of SSH's internal counter, in Section 3.9.

SSH via randomized CBC mode with random padding: SSH-$NPC. Recall that the attack against SSH-NPC involves creating a new intermediate ciphertext that would decrypt to an encoded packet that the user previously encrypted (assuming the attacker's guess was correct). With this in mind, we propose a provably secure SSH instantiation (SSH-$NPC) that uses randomized CBC mode for the underlying encryption scheme and that requires that encoded packets use random padding. We require that the random padding be chosen anew for each encryption and that the random padding occupy at least one full block of the encoded packet. This conforms to the current SSH specification since the latter allows padding up to 255 octets.

The intuition behind the security of this alternative and the reason that this alternative resists the attack in Section 3.4 is the following. Since the random padding is not sent in the clear, an attacker will not know what the random padding is and will not be able to forge a ciphertext that will decrypt to that previously encoded message (with the same random padding). Furthermore, any other attack against SSH-$NPC would translate into an attack against the underlying CBC mode encryption scheme, the underlying MAC, the encoding scheme, or the underlying block cipher.

SSH via CBC mode with CTR generated IVs: SSH-CTRIV-CBC. Instead of using CBC mode with a random IV, it is also possible to generate a "random-looking" IV by encrypting a counter with a different key; we call this alternative SSH-CTRIV-CBC. Unlike SSH-$NPC, for SSH-CTRIV-CBC we do *not* require a full block of padding and we do not require the padding to be random. The reason we do not require random padding for this alternative is because the decryptor is stateful and that any modification to an underlying CBC ciphertext will, with probability 1, change the encoded packet. This alternative is more attractive than SSH-$NPC because it does not increase the size of ciphertexts compared to SSH-IPC, but it does require one additional block cipher application compared to SSH-IPC.

SSH via CTR mode with stateful decryption: SSH-CTR. SSH-CTR uses standard CTR mode as the underlying encryption scheme with one modification: both the sender and the receiver maintain the counters themselves, rather than transmitting them as part of the ciphertexts. We refer to this variant of CTR mode as *CTR mode with stateful decryption*. We point out that this CTR mode variant offers the same level of chosen-plaintext privacy as standard CTR mode, the security of which was shown in [4]. As with SSH-CTRIV-CBC, SSH-CTR does not require additional padding and does not require the padding to be random. Furthermore, unlike SSH-$NPC and SSH-CTRIV-CBC, SSH-CTR requires the same number of block cipher invocations as SSH-IPC.

Other possibilities. There are numerous other possible fixes to the SSH BPP. Rather than enumerate all possible fixes to the SSH BPP, in Sections 3.6–3.8 we discuss how one can use our general proof techniques to prove the security of other fixes (assuming, of course, that the other fixes are indeed secure). For example, another fix of interest might be SSH-EIV-CBC, or SSH where the underlying encryption scheme is replaced by a CBC variant in which the IV is the *encipherment* of the last block of the previous ciphertext.

3.6 Definitions and the Encode-then-E&M Paradigm

Analyzing SSH via a new paradigm. An SSH ciphertext is the concatenation of the encryption and the MAC of (some encodings of) an underlying payload message. At first glance this seems to fall into the Encrypt-and-MAC method of composing an encryption scheme with a MAC. As pointed out in [10, 52] and summarized in Section 2.6.3, this particular composition method is *not* generically secure: security under standard notions of the encryption and MAC schemes used as building blocks under this composition method is not enough to guarantee the privacy of the payload. Naturally, this raises a question regarding the security of the general SSH construction.

We show here that, with an appropriate encoding method, such as the method used in SSH, an Encrypt-and-MAC-based scheme can actually be secure. In fact, our analysis models SSH more generally as an authenticated encryption scheme constructed via a paradigm we call *Encode-then-E&M*: to encrypt a message, first encode it (as SSH does), then encrypt and MAC the encoded packets. Our analysis is done in a general way in order to better ensure that the definitions and techniques we develop will be useful to the evaluators of other SSH-like schemes.

As described in Section 3.2, an SSH BPP encoded message (for encryption) consists of a packet length field, a padding length field, payload data, and padding. An encoded message (for MACing) is identical to an encoded message for encryption except that it is prepended with a 32-bit counter.

Encoding schemes. We model our use of encodings after [11] as summarized in Section 2.6.5. When we refer to encoding schemes in this chapter, we mean the type of encoding schemes that we are about to define, which share similar properties with but are different than the encodings schemes defined in Section 2.6.5.

An "encoding" scheme is an *unkeyed* transformation. We use encodings to capture the process of loading a payload message into a packet for encryption and a packet for message authentication (recall that the encoded packet that the SSH BPP encrypts is slightly different than the encoded packet that the SSH BPP MACs). Syntactically,

an *encoding scheme* $\mathcal{EC} = (\mathsf{Encode}, \mathsf{Decode})$ consists of an encoding algorithm and a decoding algorithm. The encoding algorithm Encode, which may be both randomized and stateful, takes as input a message M and returns a pair of messages (M_e, M_t). The decoding algorithm Decode, which may also be stateful but not randomized, takes as input a message M_e and returns a pair of messages (M, M_t), or $(\perp, \perp)$ on error. The following consistency requirement must be met. Consider any two messages M, M' where $|M| = |M'|$. Let $(M_e, M_t) \xleftarrow{\$} \mathsf{Encode}(M)$ for Encode in some state, and let $(M'_e, M'_t) \xleftarrow{\$} \mathsf{Encode}(M')$ for Encode is in some (possibly different) state. We require that $|M_e| = |M'_e|$ and $|M_t| = |M'_t|$. Furthermore, suppose that both Encode and Decode are in their initial states. For any sequence of messages $M^1, M^2, \ldots$ and for $i = 1, 2, \ldots$, let $(M^i_e, M^i_t) = \mathsf{Encode}(M^i)$, and then let $(m^i, m^i_t) = \mathsf{Decode}(M^i_e)$. We require that $M^i = m^i$ and that $M^i_t = m^i_t$ for all i.

Encryption schemes with stateful decryption. As in Chapter 2, a *symmetric encryption scheme* or *authenticated encryption scheme* $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ consists of three algorithms. The randomized key generation algorithm returns a key K. The encryption algorithm, which may be both randomized and stateful, takes key K and a plaintext and returns a ciphertext. Motivated by SSH, we redefine the notion of an encryption scheme to allow the decryption algorithm to be stateful, but not randomized; the decryption algorithm takes key K and a ciphertext and returns either a plaintext or a special symbol $\perp$ indicating failure. In this chapter the encryption algorithm $\mathcal{E}$ never returns $\perp$.

Consider the interaction between an encryptor and a decryptor. If at any point in time the sequence of inputs to the decryptor is not a prefix of the sequence of outputs of the encryptor, then we say that the encryption and decryption processes have become *out-of-sync* and refer to the decryption input at that point in time as the first *out-of-sync* input. The usual correctness condition, which said that if C is produced by encrypting M under K then decrypting C under K yields M, is replaced with a less stringent condition requiring only that decryption succeed when the encryption and decryption processes are in-sync. More precisely, the following must be true for any key K and

plaintexts $M_1, M_2, \ldots$. Suppose that both $\mathcal{E}_K$ and $\mathcal{D}_K$ are in their initial states. For $i = 1, 2, \ldots$, let $C_i = \mathcal{E}_K(M_i)$ and let $M_i' = \mathcal{D}_K(C_i)$. It must be that $M_i = M_i'$ for all i.

Message authentication schemes. In this chapter, we use the same definition of a message authentication scheme as in Section 2.5, but require that the tags output by the tagging algorithm all have the same length in bits.

Encode-then-E&M paradigm. Now consider an encoding scheme, and let (M_e, M_t) be the encoding of some message M. To generate a ciphertext for M using the Encode-then-E&M construction, the message M_e is encrypted with an underlying encryption scheme, the message M_t is MACed with an underlying MAC algorithm, and the resulting two values (intermediate ciphertext and MAC) are concatenated to produce the final ciphertext. The composite decryption procedure is similar except the way errors (e.g., decoding problems or tag verification failures) are handled. We take the approach used in SSH whereby, if a decryption fails, the composite decryption algorithm enters a "halting state." This approach is perhaps the most intuitive since, upon detecting a chosen-ciphertext attack, the decryption algorithm prevents all subsequent ciphertexts from being decrypted. We note, however, that this also makes the decryptor vulnerable to a denial-of-service-type attack. Construction 3.6.1 shows the Encode-then-E&M composition method in details.

Construction 3.6.1 (Encode-then-E&M.) Let $\mathcal{EC} = (\mathsf{Encode}, \mathsf{Decode})$, $\mathcal{SE} = (\mathcal{K}_e, \mathcal{E}, \mathcal{D})$, and $\mathcal{MA} = (\mathcal{K}_t, \mathcal{T}, \mathcal{V})$ respectively be encoding, encryption, and message authentication schemes with compatible message spaces (the outputs from Encode are suitable inputs to $\mathcal{E}$ and $\mathcal{T}$). Let all states initially be ε. We associate to these schemes a composite *Encode-then-E&M scheme* $\overline{\mathcal{SE}} = (\overline{\mathcal{K}}, \overline{\mathcal{E}}, \overline{\mathcal{D}})$ as follows:

Algorithm $\overline{\mathcal{K}}$

$K_e \xleftarrow{\$} \mathcal{K}_e$; $K_t \xleftarrow{\$} \mathcal{K}_t$

Return $\langle K_e, K_t \rangle$

Algorithm $\overline{\mathcal{E}}_{\langle K_e, K_t \rangle}(M)$

$(M_e, M_t) \xleftarrow{\$} \mathsf{Encode}(M)$

$\sigma \xleftarrow{\$} \mathcal{E}_{K_e}(M_e)$; $\tau \xleftarrow{\$} \mathcal{T}_{K_t}(M_t)$

$C \leftarrow \sigma \| \tau$

Return C

Algorithm $\overline{\mathcal{D}}_{\langle K_e, K_t \rangle}(C)$

If $st = \perp$ then return $\perp$

If cannot parse C then $st \leftarrow \perp$; return $\perp$

Parse C as $\sigma \| \tau$; $M_e \leftarrow \mathcal{D}_{K_e}(\sigma)$

If $M_e = \perp$ then $st \leftarrow \perp$; return $\perp$

$(M, M_t) \leftarrow \mathsf{Decode}(M_e)$

If $M = \perp$ then $st \leftarrow \perp$; return $\perp$

$v \leftarrow \mathcal{V}_{K_t}(M_t, \tau)$

If $v = 0$ then $st \leftarrow \perp$; return $\perp$

Return M

Although only $\overline{\mathcal{D}}$ explicitly maintains state in the above pseudocode, the underlying encoding, encryption, and MAC schemes may also maintain state. ▮

Security notions for encryption schemes with stateful decryption. A secure authenticated encryption scheme $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is one that preserves both privacy and integrity. The standard notion of indistinguishability (privacy) under chosen-plaintext attacks (PRIV-CPA) is as defined in Section 2.4, i.e., is unmodified even though we changed the definition of an encryption scheme to allow for a stateful decryption algorithm.

For our new notion of chosen-ciphertext privacy for stateful decryption (PRIV-SFCCA), we consider a game in which an adversary B is given access to an LR encryption oracle $\mathcal{E}_K(\mathcal{LR}(\cdot, \cdot, b))$ and a decryption oracle $\mathcal{D}_K(\cdot)$. As long as B's queries to $\mathcal{D}_K(\cdot)$ are in-sync with the responses from $\mathcal{E}_K(\mathcal{LR}(\cdot, \cdot, b))$, the decryption oracle performs the decryption (and updates its internal state) but does not return a response to B. Once B makes an out-of-sync query to $\mathcal{D}_K(\cdot)$, the decryption oracle returns the output of the decryption. We define $\mathbf{Adv}^{\text{priv-sfcca}}_{\mathcal{SE}}(B)$ as the probability that B returns 1 when $b = 1$ minus the probability that B returns 1 when $b = 0$. The new PRIV-SFCCA notion implies the previous notion of indistinguishability under chosen-ciphertext attacks, PRIV-CCA. Note that, without allowing an adversary to query the decryption oracle with

in-sync ciphertexts (e.g., in the standard PRIV-CCA setting), we would not be able to model attacks in which the adversary attacks a stateful decryptor after the latter had decrypted a number of legitimate ciphertexts (perhaps because of some weakness related to the state of the decryptor at that time). A more formal presentation of this new definition follows.

Definition 3.6.2 (Privacy for symmetric encryption schemes with stateful decryption.) Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme. Let A_{sfcca} be an adversary that has access to a left-or-right encryption oracle $\mathcal{E}_K(\mathcal{LR}(\cdot,\cdot,b))$ and a decryption oracle $\mathcal{D}_K(\cdot)$. The adversary returns a bit. Consider the experiments below, where $b \in \{0,1\}$ is a bit.

Experiment $\mathbf{Exp}^{\text{priv-sfcca-b}}_{\mathcal{SE}}(A_{\text{sfcca}})$
  $K \xleftarrow{\$} \mathcal{K}$; $i \leftarrow 0$; $j \leftarrow 0$; $\mathsf{phase} \leftarrow 0$
  Run $A^{\mathcal{E}_K(\mathcal{LR}(\cdot,\cdot,b)),\mathcal{D}_K(\cdot)}_{\text{sfcca}}$
    Reply to $\mathcal{E}_K(\mathcal{LR}(M_0, M_1, b))$ queries as follows:
      $i \leftarrow i + 1$; $C_i \xleftarrow{\$} \mathcal{E}_K(M_b)$; $A_{\text{sfcca}} \Leftarrow C_i$
    Reply to $\mathcal{D}_K(C)$ queries as follows:
      $j \leftarrow j + 1$; $M \leftarrow \mathcal{D}_K(C)$
      If $j > i$ or $C \neq C_j$ then $\mathsf{phase} \leftarrow 1$
      If $\mathsf{phase} = 1$ then $A_{\text{sfcca}} \Leftarrow M$
  Until A_{sfcca} returns a bit d
  Return d

We require that, for all queries (M_0, M_1) to $\mathcal{E}_K(\mathcal{LR}(\cdot,\cdot,b))$, $|M_0| = |M_1|$. We define the PRIV-SFCCA-advantage, of the adversary as

$$\begin{aligned}\mathbf{Adv}^{\text{priv-sfcca}}_{\mathcal{SE}}(A_{\text{sfcca}}) \;&=\; \Pr\left[\,\mathbf{Exp}^{\text{priv-sfcca-1}}_{\mathcal{SE}}(A_{\text{sfcca}}) = 1\,\right] \\ &\quad -\; \Pr\left[\,\mathbf{Exp}^{\text{priv-sfcca-0}}_{\mathcal{SE}}(A_{\text{sfcca}}) = 1\,\right] .\end{aligned}$$

In the concrete setting [6], we say that $\mathcal{SE}$ is PRIV-SFCCA-secure if $\mathbf{Adv}^{\text{priv-sfcca}}_{\mathcal{SE}}(A_{\text{sfcca}})$ is small for all adversaries A_{sfcca} using reasonable resources. ∎

Section 2.6 gives the standard notion for integrity of plaintexts (AUTHP) and integrity of ciphertexts (AUTHC) from [10], both of which still apply to symmetric encryption schemes with stateful decryption algorithms. For our new notion of integrity of ciphertexts for stateful decryption (AUTHSF), we again consider a game in which an adversary F_{sf} is given access to the two oracles $\mathcal{E}_K(\cdot)$ and $\mathcal{D}^*_K(\cdot)$. We define $\mathbf{Adv}^{\text{authsf}}_{\mathcal{SE}}(F_{\text{sf}})$ as the probability that F_{sf} can generate a ciphertext C such that $\mathcal{D}^*_K(C) = 1$ and C is an out-of-sync query. The new notion of AUTHSF implies the previous notion of integrity of ciphertexts, AUTHC, as well as security against replay and out-of-order delivery attacks. A more formal presentation of the definitions follows.

Definition 3.6.3 (Stateful ciphertext integrity.) Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme. Let F_{sf} be an adversary with access to an encryption oracle $\mathcal{E}_K(\cdot)$ and a decryption-verification oracle $\mathcal{D}^*_K(\cdot)$. The decryption-verification oracle invokes $\mathcal{D}_K(C)$ and returns 1 if $\mathcal{D}_K(\cdot) \neq \perp$ and 0 otherwise. Consider the experiment below.

Experiment $\mathbf{Exp}^{\text{authsf}}_{\mathcal{SE}}(F_{\text{sf}})$
  $K \xleftarrow{\$} \mathcal{K}$; $i \leftarrow 0$; $j \leftarrow 0$; $\mathsf{phase} \leftarrow 0$
  Run $A^{\mathcal{E}_K(\cdot),\mathcal{D}^*_K(\cdot)}_{\text{ctxt}}$
    Reply to $\mathcal{E}_K(M)$ queries as follows: $i \leftarrow i + 1$; $C_i \xleftarrow{\$} \mathcal{E}_K(M)$; $F_{\text{sf}} \Leftarrow C_i$
    Reply to $\mathcal{D}^*_K(C)$ queries as follows:
      $j \leftarrow j + 1$; $M \leftarrow \mathcal{D}_K(C)$
      If $j > i$ or $C \neq C_j$ then $\mathsf{phase} \leftarrow 1$
      If $M \neq \perp$ and $\mathsf{phase} = 1$ then return 1
      If $M \neq \perp$ then $F_{\text{sf}} \Leftarrow 1$ else $F_{\text{sf}} \Leftarrow 0$
  Until F_{sf} halts
  Return 0

We define the AUTHSF-advantage of the adversary F_{sf} in attacking the *stateful ciphertext integrity* of the scheme as

$$\mathbf{Adv}^{\text{authsf}}_{\mathcal{SE}}(F_{\text{sf}}) \;=\; \Pr\left[\, \mathbf{Exp}^{\text{authsf}}_{\mathcal{SE}}(F_{\text{sf}}) = 1 \,\right] .$$

In the concrete setting [6], we say that $\mathcal{SE}$ *preserves integrity of stateful ciphertexts* (AUTHSF-secure) if the advantage $\mathbf{Adv}^{\text{authsf}}_{\mathcal{SE}}(F_{\text{sf}})$ is small for all forgers F_{sf} using reasonable resources. ▌

The following proposition states that, if an authenticated encryption scheme is indistinguishable under chosen-plaintexts attacks and if the scheme meets our strong definition of integrity of ciphertexts, then the scheme will meet our strong definition of indistinguishability under chosen-ciphertext attacks. It is similar to the results in [10] and [47], restated in Section 2.6.1, which show that the standard PRIV-CPA and the standard AUTHC notions imply the standard PRIV-CCA notion.

Proposition 3.6.4 Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an authenticated encryption scheme. Given any PRIV-SFCCA adversary A, we can construct an AUTHSF adversary I and an PRIV-CPA adversary B such that

$$\mathbf{Adv}^{\text{priv-sfcca}}_{\mathcal{SE}}(A) \leq 2 \cdot \mathbf{Adv}^{\text{authsf}}_{\mathcal{SE}}(I) + \mathbf{Adv}^{\text{priv-cpa}}_{\mathcal{SE}}(B)$$

and I and B use the same resources as A. ▌

Proof of of Proposition 3.6.4: Our proof is modeled after the proof of a similar property in [10]. Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme, and let A be any PRIV-SFCCA adversary against $\mathcal{SE}$. We associate to A a PRIV-CPA adversary B and an AUTHSF adversary I. The adversary B runs A almost exactly as in $\mathbf{Exp}^{\text{priv-sfcca-b}}_{\mathcal{SE}}(A)$ where b is B's LR encryption oracle bit. The only exception is that B return $\perp$ to A if A submits an out-of-sync decryption query. Then, B outputs what A outputs. Similarly, I runs A almost exactly as in $\mathbf{Exp}^{\text{priv-sfcca-A}}_{\mathcal{SE}}(b)$ where b is a bit that I chooses at random. The only exception is that, when A successfully submits an out-of-sync decryption query, the adversary I terminates.

Let $\Pr_1[\,\cdot\,]$ denote the probability over $\mathbf{Exp}^{\text{priv-sfcca-b}}_{\mathcal{SE}}(A)$ and a random choice for $b \in \{0,1\}$, and let b' denote the output of A in these experiments. Let $\Pr_2[\,\cdot\,]$ denote the probability in $\mathbf{Exp}^{\text{authsf}}_{\mathcal{SE}}(I)$. Let $\Pr_3[\,\cdot\,]$ denote the probability over $\mathbf{Exp}^{\text{priv-cpa-c}}_{\mathcal{SE}}(B)$ where c is randomly selected from $\{0,1\}$ and let c' be the bit B returns. Let E denote

the event that A makes at least one query to a phase 1 decryption oracle that would successfully decrypt. Note that

$$\Pr_1[\, b' = b \wedge E \,] \leq \Pr_1[\, E \,] \leq \mathbf{Adv}^{\text{authsf}}_{\mathcal{SE}}(I)$$

since, prior to E occurring, $\mathbf{Exp}^{\text{authsf}}_{\mathcal{SE}}(I)$ runs A exactly as in $\mathbf{Exp}^{\text{priv-sfcca-b}}_{\mathcal{SE}}(A)$ for a random b and, once E occurs, I succeeds in forging a ciphertext. Also,

$$\begin{aligned}
&\Pr_1\left[\, b' = b \wedge \overline{E} \,\right] \leq \Pr_3[\, c' = c \,] \\
&\quad = \frac{1}{2} \cdot \Pr\left[\, \mathbf{Exp}^{\text{priv-cpa-1}}_{\mathcal{SE}}(B) = 1 \,\right] + \frac{1}{2} \cdot \left(1 - \Pr\left[\, \mathbf{Exp}^{\text{priv-cpa-0}}_{\mathcal{SE}}(B) = 1 \,\right]\right) \\
&\quad = \frac{1}{2}\mathbf{Adv}^{\text{priv-cpa}}_{\mathcal{SE}}(B) + \frac{1}{2}
\end{aligned}$$

since whenever A does not cause event E to occur, A's view when run by B is equivalent to its view when run in $\mathbf{Exp}^{\text{priv-sfcca-b}}_{\mathcal{SE}}(A)$. Consequently,

$$\begin{aligned}
\frac{1}{2}\mathbf{Adv}^{\text{priv-sfcca}}_{\mathcal{SE}}(A) + \frac{1}{2} &= \Pr_1[\, b' = b \,] \\
&= \Pr_1[\, b' = b \wedge E \,] + \Pr_1\left[\, b' = b \wedge \overline{E} \,\right] \\
&\leq \mathbf{Adv}^{\text{authsf}}_{\mathcal{SE}}(I) + \frac{1}{2}\mathbf{Adv}^{\text{priv-cpa}}_{\mathcal{SE}}(B) + \frac{1}{2}\,.
\end{aligned}$$

The adversaries B and I use the same resources as A except that B does not perform any chosen-ciphertext queries to a decryption oracle. ∎

Collision resistance of encoding schemes. The security of a composite Encode-then-E&M construction depends on properties of the underlying encoding, encryption, and MAC schemes. In addition to the standard assumptions of indistinguishability under chosen-plaintext attacks of the encryption scheme and unforgeability and pseudorandomness of the MAC scheme, we require *collision resistance* of the encoding scheme. We motivate this notion as follows. Consider an integrity adversary against a composite Encode-then-E&M scheme. If the adversary can find two different messages that encode (or decode) to the same input for the underlying MAC, then the adversary may be able to compromise the integrity of the composite scheme. Consider now an indistinguishability adversary against the composite scheme. As long as the adversary does not

generate two inputs for the underlying MAC that collide, the underlying MAC should not leak information about the plaintext. The following describes the notions of collision resistance for encoding schemes.

An adversary A who is mounting a chosen-plaintext attack against an encoding scheme $\mathcal{EC} = (\mathsf{Encode}, \mathsf{Decode})$ is given access to an encoding oracle $\mathsf{Encode}(\cdot)$. If A can make the encoding oracle output two pairs that collide on their second components (i.e., the M_t's), then A wins. We allow A to repeatedly query the encoding oracle with the same input. Similarly, an adversary B mounting a chosen-ciphertext attack against $\mathcal{EC}$ is given access to both an encoding oracle and a decoding oracle $\mathsf{Decode}(\cdot)$. If B can cause a collision in the second components of the outputs of $\mathsf{Encode}(\cdot)$, $\mathsf{Decode}(\cdot)$, or both, then it wins. We exclude the cases where B uses the two oracles in a trivial way to obtain collisions (e.g., submitting a query to $\mathsf{Encode}(\cdot)$ and then immediately submitting the first component of the result, namely M_e, to $\mathsf{Decode}(\cdot)$). We refer to the advantages of the adversaries in these two settings as $\mathbf{Adv}^{\text{coll-cpa}}_{\mathcal{EC}}(A)$ and $\mathbf{Adv}^{\text{coll-cca}}_{\mathcal{EC}}(B)$, respectively. All encoding schemes with deterministic and stateless encoding algorithms are insecure under chosen-plaintext collision attacks. Furthermore, all encoding schemes with stateless decoding algorithms are insecure under chosen-ciphertext collision attacks. A more formal presentation of the definitions follows.

Definition 3.6.5 (Collision resistance.) Let $\mathcal{EC} = (\mathsf{Encode}, \mathsf{Decode})$ be a encoding scheme. Let A_{cpa} be an adversary with access to an encoding oracle and let A_{cca} be an adversary with access to an encoding oracle $\mathsf{Encode}(\cdot)$ and a decoding oracle $\mathsf{Decode}(\cdot)$. Let M^i denote an adversary's i-th encoding query and let (M^i_e, M^i_t) denote the response for that query. Let m^i_e denote A_{cca}'s i-th decoding query and let (m^i, m^i_t) denote the response for that query. Consider the following experiments:

Experiment $\mathbf{Exp}^{\text{coll-cpa}}_{\mathcal{EC}}(A_{\text{cpa}})$
  Run $A^{\mathsf{Encode}(\cdot)}_{\text{cpa}}$
  If $A^{\mathsf{Encode}(\cdot)}_{\text{cpa}}$ makes two queries M^i, M^j to $\mathsf{Encode}(\cdot)$
    such that $i \neq j$ and $M^i_t = M^j_t$ then return 1 else return 0

Experiment $\mathbf{Exp}^{\text{coll-cca}}_{\mathcal{EC}}(A_{\text{cca}})$

Run $A_{\text{cca}}^{\mathsf{Encode}(\cdot),\mathsf{Decode}(\cdot)}$

If one of the following occurs:

— A_{cca} makes two queries M^i, M^j to $\mathsf{Encode}(\cdot)$
such that $i \neq j$ and $M^i_t = M^j_t$

— A_{cca} makes two queries m^i_e, m^j_e to $\mathsf{Decode}(\cdot)$
such that $i \neq j$, $m^i_t \neq \perp$, and $m^i_t = m^j_t$

— A_{cca} makes a query M^i to $\mathsf{Encode}(\cdot)$ and a query m^j_e to $\mathsf{Decode}(\cdot)$
such that ($i \neq j$ or $M^i \neq m^j$ or $M^i_e \neq m^j_e$) and $M^i_t = m^j_t$

then return 1 else return 0

We respectively define the COLL-CPA- and COLL-CCA-advantages of the adversaries A_{cpa} and A_{cca} in finding a collision as

$$\begin{aligned}\mathbf{Adv}^{\text{coll-cpa}}_{\mathcal{EC}}(A_{\text{cpa}}) &= \Pr\left[\,\mathbf{Exp}^{\text{coll-cpa}}_{\mathcal{EC}}(A_{\text{cpa}}) = 1\,\right]\\ \mathbf{Adv}^{\text{coll-cca}}_{\mathcal{EC}}(A_{\text{cca}}) &= \Pr\left[\,\mathbf{Exp}^{\text{coll-cca}}_{\mathcal{EC}}(A_{\text{cca}}) = 1\,\right]\ .\end{aligned}$$

In the concrete setting [6], we say that $\mathcal{EC}$ meets the respective definition of *collision resistance*, i.e., are COLL-CPA- and COLL-CCA-secure, if the advantages $\mathbf{Adv}^{\text{coll-cpa}}_{\mathcal{EC}}(A_{\text{cpa}})$ and $\mathbf{Adv}^{\text{coll-cca}}_{\mathcal{EC}}(A_{\text{cca}})$ are small for all adversaries A_{cpa} and A_{cca} using reasonable resources. ∎

3.7 General Security Results for the Encode-then-E&M Paradigm

Since our analysis models SSH more generally as an authenticated encryption scheme constructed via the Encode-then-E&M paradigm, we first present here general results for the Encode-then-E&M composition method. In Section 3.8 we build upon these results and prove additional properties about our proposed fixes to SSH. The results in this section will also be useful to the evaluators of other Encode-then-E&M constructions.

3.7.1 Chosen-Plaintext Privacy

To build an authenticated encryption scheme that provides chosen-plaintext privacy via the Encode-the-E&M paradigm, it is enough to use a PRIV-CPA-secure encryption scheme, a pseudorandom MAC, and a COLL-CPA-secure encoding scheme as building blocks. The following theorem states this result more formally. We defer the proof of Theorem 3.7.1 to Section 3.7.3. Recall again that the basic Encrypt-and-MAC paradigm does not provide privacy under chosen-plaintext attacks when the underlying MAC is stateless and deterministic.

Theorem 3.7.1 (Privacy for Encode-then-E&M with respect to chosen-plaintext attacks.) Let $\mathcal{SE}$, $\mathcal{MA}$, and $\mathcal{EC}$ respectively be an encryption, a message authentication, and an encoding scheme. Let $\overline{\mathcal{SE}}$ be the encryption scheme associated to them as per Construction 3.6.1. Then, given any PRIV-CPA adversary S against $\overline{\mathcal{SE}}$, we can construct adversaries A, D, and C such that

$$\mathbf{Adv}^{\text{priv-cpa}}_{\overline{\mathcal{SE}}}(S) \leq \mathbf{Adv}^{\text{priv-cpa}}_{\mathcal{SE}}(A) + 2 \cdot \mathbf{Adv}^{\text{prf}}_{\mathcal{MA}}(D) + 2 \cdot \mathbf{Adv}^{\text{coll-cpa}}_{\mathcal{EC}}(C) \ .$$

Furthermore, $A, D,$ and C use the same resources as S except that A's and D's inputs to their respective oracles may be of different lengths than those of S (due to the encoding). ∎

3.7.2 Integrity of Plaintexts

The following theorem states that the composed scheme provides plaintext integrity if the underlying MAC is unforgeable[1] and if the underlying encoding scheme is collision-resistant against chosen-ciphertext attacks. We need more than chosen-plaintext collision resistance of the underlying encoding scheme here because an adversary is allowed to submit ciphertext queries when mounting an integrity attack. We

[1] Although the theorem statement refers to strong unforgeability [10], weak unforgeability [6] of the underlying MAC scheme is actually sufficient here since the COLL-CCA property of the underlying encoding scheme ensures that inputs to the MAC algorithm will not collide.

remark that the combination of PRIV-CPA and AUTHP does not, however, imply our notion of privacy under chosen-ciphertext attacks, as exemplified by the reaction attack in Section 3.4 and the fact that the construction in Section 3.4 is both PRIV-CPA- and AUTHP-secure; we consider how to achieve our chosen-ciphertext privacy notion, via our integrity of ciphertexts notion, in Section 3.8.

Theorem 3.7.2 (Integrity of plaintexts for Encode-then-E&M.) Let $\mathcal{SE}$ be a symmetric encryption scheme, let $\mathcal{MA}$ be a message authentication scheme, and let $\mathcal{EC}$ be an encoding scheme. Let $\overline{\mathcal{SE}}$ be the encryption scheme associated to them as per Construction 3.6.1. Then, given any AUTHP adversary A against $\overline{\mathcal{SE}}$, we can construct adversaries F and C such that

$$\mathbf{Adv}^{\text{authp}}_{\overline{\mathcal{SE}}}(A) \leq \mathbf{Adv}^{\text{uf}}_{\mathcal{MA}}(F) + \mathbf{Adv}^{\text{coll-cca}}_{\mathcal{EC}}(C) \ .$$

Furthermore, F and C use the same resources as A except that F's messages to its tagging and tag verification oracles may be slightly larger than A's encryption queries (due to the encoding) and that C's messages to its decoding oracle may have different lengths than A's decryption queries. ∎

Proof of of Theorem 3.7.2: Let $\overline{\mathcal{SE}} = (\overline{\mathcal{K}}, \overline{\mathcal{E}}, \overline{\mathcal{D}})$ be the composite encryption scheme constructed via Construction 3.6.1 from the encryption scheme $\mathcal{SE} = (\mathcal{K}_e, \mathcal{E}, \mathcal{D})$, the MAC scheme $\mathcal{MA} = (\mathcal{K}_t, \mathcal{T}, \mathcal{V})$, and the encoding scheme $\mathcal{EC} = (\mathsf{Encode}, \mathsf{Decode})$. Assume we have an adversary A attacking the integrity of plaintexts of $\overline{\mathcal{SE}}$. We associate to A two adversaries: a forger F breaking the unforgeability of $\mathcal{MA}$ and a collision finder C breaking the collision resistance of $\mathcal{EC}$ such that

$$\mathbf{Adv}^{\text{authp}}_{\overline{\mathcal{SE}}}(A) \leq \mathbf{Adv}^{\text{uf}}_{\mathcal{MA}}(F) + \mathbf{Adv}^{\text{coll-cca}}_{\mathcal{EC}}(C) \ . \tag{3.1}$$

The forger F and the collision finder C are simple. The forger F uses $\mathcal{K}_e$ to generate an encryption key and uses the encryption key and its tagging oracle to answer A's queries in a straight-forward manner. In particular, it follows Construction 3.6.1. Similarly, the collision finder C uses the same approach. This ensures that A is executed in the same environment as that in $\mathbf{Exp}^{\text{authp}}_{\overline{\mathcal{SE}}}(A)$.

Let $\Pr_1[\,\cdot\,]$, $\Pr_2[\,\cdot\,]$, and $\Pr_3[\,\cdot\,]$ respectively denote the probabilities associated with the experiments $\mathbf{Exp}^{\text{authp}}_{\overline{\mathcal{SE}}}(A)$, $\mathbf{Exp}^{\text{uf}}_{\mathcal{MA}}(F)$, and $\mathbf{Exp}^{\text{coll-cca}}_{\mathcal{EC}}(C)$. Let E denote the event that A makes a query that would cause C to succeed in finding a collision. Then, by the definition of E,

$$\Pr_1[\,E\,] = \Pr_3\left[\,\mathbf{Exp}^{\text{coll-cca}}_{\mathcal{EC}}(C) = 1\,\right] .$$

Furthermore,

$$\Pr_1\left[\,\mathbf{Exp}^{\text{authp}}_{\overline{\mathcal{SE}}}(A) = 1 \wedge \overline{E}\,\right] \leq \Pr_2\left[\,\mathbf{Exp}^{\text{uf}}_{\mathcal{MA}}(F) = 1\,\right]$$

since $\overline{E}$ implies that the verification request that caused A to succeed must have produced (through the decoding) a previously unseen tagging message M_t (thereby allowing F to succeed). Consequently,

$$\begin{aligned}
&\Pr_1\left[\,\mathbf{Exp}^{\text{authp}}_{\overline{\mathcal{SE}}}(A) = 1\,\right] \\
&\qquad = \Pr_1\left[\,\mathbf{Exp}^{\text{authp}}_{\overline{\mathcal{SE}}}(A) = 1 \wedge \overline{E}\,\right] + \Pr_1\left[\,\mathbf{Exp}^{\text{authp}}_{\overline{\mathcal{SE}}}(A) = 1 \wedge E\,\right] \\
&\qquad \leq \Pr_2\left[\,\mathbf{Exp}^{\text{uf}}_{\mathcal{MA}}(F) = 1\,\right] + \Pr_3\left[\,\mathbf{Exp}^{\text{coll-cca}}_{\mathcal{EC}}(C) = 1\,\right]
\end{aligned}$$

and Equation 3.1 follows. Adversaries F and A use equivalent resources except that F's messages to its oracles may be slightly larger due to the encoding. Adversaries C and A also use equivalent resources except that C's message to its oracle may not be the exactly the same size as A's decryption-verification queries, although they are polynomially related. ∎

3.7.3 Proof of Theorem 3.7.1

We now prove Theorem 3.7.1. One notable feature of the proof is that it actually uses a weaker property than pseudorandomness for the underlying MAC. The said property is the following.

Distinct plaintext privacy of message authentication schemes. Let $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ be a message authentication scheme. The notion of PRIV-DCPA for $\mathcal{MA}$ is based

on the PRIV-CPA notion for encryption. For a bit b and a key K, let $\mathcal{T}_K(\mathcal{LR}(\cdot,\cdot,b))$ denote the *LR tag oracle* which, given equal-length plaintexts M_0, M_1, returns $\mathcal{T}_K(M_b)$. We stress that the LR tag oracle returns only the tag and *not* the message-tag pair $M_b \| \mathcal{T}_K(M_b)$. The PRIV-DCPA notion is defined as follows.

Definition 3.7.3 (Privacy against distinct chosen-plaintext attacks.) Let $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ be a message authentication scheme. Let $b \in \{0,1\}$. Let A be an adversary that has access to an oracle $\mathcal{T}_K(\mathcal{LR}(\cdot,\cdot,b))$. Consider the following experiment:

Experiment $\mathbf{Exp}^{\text{priv-dcpa-b}}_{\mathcal{MA}}(A_{\text{cpa}})$
  $K \xleftarrow{\$} \mathcal{K}$
  Run $A_{\text{cpa}}^{\mathcal{T}_K(\mathcal{LR}(\cdot,\cdot,b))}$
    Reply to $\mathcal{T}_K(\mathcal{LR}(M_0, M_1, b))$ queries as follows:
      $C \xleftarrow{\$} \mathcal{T}_K(M_b)$; $A_{\text{cpa}} \Leftarrow C$
  Until A_{cpa} returns a bit d
  Return d

Above it is mandated that all left messages of A's queries be unique and that all right messages of A's queries be unique. We define the PRIV-DCPA-advantage of A via

$$\mathbf{Adv}^{\text{priv-dcpa}}_{\mathcal{MA}}(A) = \Pr\left[\mathbf{Exp}^{\text{priv-dcpa-1}}_{\mathcal{MA}}(A) = 1\right] - \Pr\left[\mathbf{Exp}^{\text{priv-dcpa-0}}_{\mathcal{MA}}(A) = 1\right] .$$

In the concrete setting [6], we say that $\mathcal{MA}$ is *privacy-preserving* under *distinct chosen plaintext attacks* (PRIV-DCPA-secure) if $\mathbf{Adv}^{\text{priv-dcpa}}_{\mathcal{MA}}(A)$ is small for all adversaries A using reasonable resources. ▮

The following theorem relates the distinct plaintext privacy and pseudorandomness notions.

Theorem 3.7.4 (Relation between IND-DCPA and PRF.) Let $\mathcal{MA}$ be a message authentication scheme. Then, given any PRIV-DCPA adversary A against $\mathcal{MA}$, we can construct a distinguisher D against $\mathcal{MA}$ such that

$$\mathbf{Adv}^{\text{priv-dcpa}}_{\mathcal{MA}}(A) \leq 2 \cdot \mathbf{Adv}^{\text{prf}}_{\mathcal{MA}}(D)$$

Furthermore, D uses the same resources of A. ▮

This theorem implies that if $\mathcal{MA}$ is secure as a PRF, as is expected of many MACs [6], then it will also be PRIV-DCPA-secure. The theorem is easy to verify; we omit the proof.

Theorem 3.7.1 follows directly from Theorem 3.7.4 above and Lemma 3.7.5 presented below. We therefore turn our attention to Lemma 3.7.5 below. Throughout, we let $\mathsf{Encode}^*(\cdot,\cdot)$ and $\mathsf{Decode}^*(\cdot,\cdot)$ denote the encoding algorithms $\mathsf{Encode}(\cdot)$ and $\mathsf{Decode}(\cdot)$ except that they explicitly take a state as part of the input and return a new state as part of the output.

Lemma 3.7.5 Let $\mathcal{SE} = (\mathcal{K}_e, \mathcal{E}, \mathcal{D})$ be an encryption scheme, let $\mathcal{MA} = (\mathcal{K}_t, \mathcal{T}, \mathcal{V})$ be a message authentication scheme, and let $\mathcal{EC} = (\mathsf{Encode}, \mathsf{Decode})$ be an encoding scheme. Let $\overline{\mathcal{SE}}$ be the encryption scheme associated to them as per Construction 3.6.1. Then, given any PRIV-CPA adversary S against $\overline{\mathcal{SE}}$, we can construct a PRIV-CPA adversary A against $\mathcal{SE}$, a PRIV-DCPA adversary B against $\mathcal{MA}$, and a collision finder C such that

$$\mathbf{Adv}^{\text{priv-cpa}}_{\overline{\mathcal{SE}}}(S) \le \mathbf{Adv}^{\text{priv-cpa}}_{\mathcal{SE}}(A) + \mathbf{Adv}^{\text{priv-dcpa}}_{\mathcal{MA}}(B) + 2 \cdot \mathbf{Adv}^{\text{coll-cpa}}_{\mathcal{EC}}(C)\ .$$

Furthermore, $A, B,$ and C use the same resources as S except that A's and B's inputs to their respective oracles may be slightly larger than those of S (due to the encoding). ∎

Proof of of Lemma 3.7.5: Let S denote a PRIV-CPA adversary that has access to an $\overline{\mathcal{E}}_K(\mathcal{LR}(\cdot,\cdot,b))$ oracle, $b \in \{0,1\}$. Let $x \in \{1,2,3\}$. We define three experiments associated with S as follows.

Experiment $\mathbf{ExpH}_x$

$K_e \xleftarrow{\$} \mathcal{K}_e\ ;\ K_t \xleftarrow{\$} \mathcal{K}_t\ ;\ st_0 \leftarrow \varepsilon\ ;\ st_1 \leftarrow \varepsilon$

Run S replying to its oracle query (M_0, M_1) as follows:

$(M_{e,0}, M_{t,0}, st_0) \xleftarrow{\$} \mathsf{Encode}^*(M_0, st_0)$

$(M_{e,1}, M_{t,1}, st_1) \xleftarrow{\$} \mathsf{Encode}^*(M_1, st_1)$

Switch (x):

Case $x = 1$: $\sigma \xleftarrow{\$} \mathcal{E}_{K_e}(M_{e,1})\ ;\ \tau \xleftarrow{\$} \mathcal{T}_{K_t}(M_{t,1})$

Case $x = 2$: $\sigma \xleftarrow{\$} \mathcal{E}_{K_e}(M_{e,0})\ ;\ \tau \xleftarrow{\$} \mathcal{T}_{K_t}(M_{t,1})$

Case $x = 3$: $\sigma \xleftarrow{\$} \mathcal{E}_{K_e}(M_{e,0})$; $\tau \xleftarrow{\$} \mathcal{T}_{K_t}(M_{t,0})$

$S \Leftarrow \sigma \| \tau$

Until S halts and returns a bit b

Return b.

Let $P_x \stackrel{\text{def}}{=} \Pr[\, \mathbf{ExpH}_x = 1 \,]$ denote the probability that experiment $\mathbf{ExpH}_x$ returns 1, for $x \in \{1, 2, 3\}$. By the definition of $\mathbf{Adv}^{\text{priv-cpa}}_{\mathcal{SE}}(S)$, we have

$$\mathbf{Adv}^{\text{priv-cpa}}_{\mathcal{SE}}(S) = P_1 - P_3 = (P_1 - P_2) + (P_2 - P_3)\ . \tag{3.2}$$

Given S, we can construct three new adversaries A, B, and C such that the following lemmas hold and the new adversaries use the resources specified in the statement of Lemma 3.7.5.

Lemma 3.7.6 $P_1 - P_2 \leq \mathbf{Adv}^{\text{priv-cpa}}_{\mathcal{SE}}(A)$.

Lemma 3.7.7 $P_2 - P_3 \leq \mathbf{Adv}^{\text{priv-dcpa}}_{\mathcal{MA}}(B) + 2 \cdot \mathbf{Adv}^{\text{coll-cpa}}_{\mathcal{EC}}(C)$.

Equation 3.2 and the above lemmas imply Lemma 3.7.5. ∎

Proof of of Lemma 3.7.6: We construct an adversary A breaking privacy of the underlying encryption scheme $\mathcal{SE} = (\mathcal{K}_e, \mathcal{E}, \mathcal{D})$ using the adversary S below.

Adversary $A^{\mathcal{E}_K(\mathcal{LR}(\cdot,\cdot,b))}$

$K_t \xleftarrow{\$} \mathcal{K}_t$; $st_0 \leftarrow \varepsilon$; $st_1 \leftarrow \varepsilon$

Run S replying to its oracle query (M_0, M_1) as follows:

$(M_{e,0}, M_{t,0}, st_0) \xleftarrow{\$} \mathsf{Encode}^*(M_0, st_0)$

$(M_{e,1}, M_{t,1}, st_1) \xleftarrow{\$} \mathsf{Encode}^*(M_1, st_1)$

$\sigma \xleftarrow{\$} \mathcal{E}_K(\mathcal{LR}(M_{e,0}, M_{e,1}, b))$; $\tau \xleftarrow{\$} \mathcal{T}_{K_t}(M_{t,1})$

$S \Leftarrow \sigma \| \tau$

Until S halts and returns a bit b'

Return b'.

If $b = 1$, the adversary A simulates S in the exact same environment as that of $\mathbf{ExpH}_1$. Similarly, if $b = 0$, the adversary A simulates S in the exact same environment as that of $\mathbf{ExpH}_2$. Thus,

$$\begin{aligned} P_1 - P_2 &= \Pr\left[\mathbf{Exp}^{\text{priv-cpa-1}}_{\mathcal{SE}}(A) = 1\right] - \Pr\left[\mathbf{Exp}^{\text{priv-cpa-0}}_{\mathcal{SE}}(A) = 1\right] \\ &= \mathbf{Adv}^{\text{priv-cpa}}_{\mathcal{SE}}(A)\ . \end{aligned}$$

The adversary A uses the same resources as S except that, due to the encoding, the queries that A makes to its oracle may be slightly larger than the queries that S makes to its oracle. Also, A performs two encodings for each query that S makes and, thus, its running time is polynomially larger than that of S. Recall the standard convention that running time of an adversary is measured with respect to the entire experiment in which it runs. Hence, Lemma 3.7.6 follows. ∎

Proof of of Lemma 3.7.7: Given S, we can construct an adversary B that can break the distinct chosen-plaintexts privacy of the underlying MAC scheme $\mathcal{MA} = (\mathcal{K}_t, \mathcal{T}, \mathcal{V})$ and an adversary C that can break the collision resistance of the underlying encoding scheme $\mathcal{EC} = (\mathsf{Encode}, \mathsf{Decode})$. These adversaries are defined in below.

Adversary $B^{\mathcal{T}_K(\mathcal{LR}(\cdot,\cdot,b))}$

$K_e \xleftarrow{\$} \mathcal{K}_e\ ;\ st_0 \leftarrow \varepsilon\ ;\ st_1 \leftarrow \varepsilon$

Run S replying to its ith oracle query (M_0^i, M_1^i) as follows:

$(M_{e,0}^i, M_{t,0}^i, st_0) \xleftarrow{\$} \mathsf{Encode}^*(M_0^i, st_0)$

$(M_{e,1}^i, M_{t,1}^i, st_1) \xleftarrow{\$} \mathsf{Encode}^*(M_1^i, st_1)$

If $M_{t,0}^i \in \{M_{t,0}^1, \ldots, M_{t,0}^{i-1}\}$ or $M_{t,1}^i \in \{M_{t,1}^1, \ldots, M_{t,1}^{i-1}\}$ then return 0

$\sigma \xleftarrow{\$} \mathcal{E}_{K_e}(M_{e,0}^i)$

$\tau \xleftarrow{\$} \mathcal{T}_K(\mathcal{LR}(M_{t,0}^i, M_{t,1}^i, b))$

$S \Leftarrow \sigma \| \tau$

Until S halts and returns a bit b

Return b

Adversary $C^{\mathsf{Encode}(\cdot)}$

$K_e \xleftarrow{\$} \mathcal{K}_e \; ; \; K_t \xleftarrow{\$} \mathcal{K}_t \; ; \; st_n \leftarrow \varepsilon$

$d \xleftarrow{\$} \{0,1\} \; ; \; c \leftarrow \overline{d}$

Run S replying to its oracle query (M_0, M_1) as follows:

$(M_{e,d}, M_{t,d}) \xleftarrow{\$} \mathsf{Encode}(M_d)$

$(M_{e,c}, M_{t,c}, st_n) \xleftarrow{\$} \mathsf{Encode}^*(M_c, st_n)$

$\sigma \xleftarrow{\$} \mathcal{E}_{K_e}(M_{e,0})$

$\tau \xleftarrow{\$} \mathcal{T}_{K_t}(M_{t,1})$

$S \Leftarrow \sigma \| \tau$

Until S halts and returns a bit b

Let $\Pr_2[\,\cdot\,]$ and $\Pr_3[\,\cdot\,]$ denote the probabilities associated with the experiments $\mathbf{ExpH}_2$ and $\mathbf{ExpH}_3$, respectively. Let E_2 denote an event that there exists at least one collision among the $M_{t,0}$'s or among the $M_{t,1}$'s in $\mathbf{ExpH}_2$. Let E_3 denote an event that there exists at least one collision among the $M_{t,0}$'s or among the $M_{t,1}$'s in $\mathbf{ExpH}_3$. We make the following claims.

Claim 3.7.8 $\Pr_2[\,E_2\,] \leq 2 \cdot \mathbf{Adv}^{\text{coll-cpa}}_{\mathcal{EC}}(C)$.

Claim 3.7.9 $\Pr_2\left[\,\mathbf{ExpH}_2 = 1 \wedge \overline{E_2}\,\right] - \Pr_3\left[\,\mathbf{ExpH}_3 = 1 \wedge \overline{E_3}\,\right] = \mathbf{Adv}^{\text{priv-dcpa}}_{\mathcal{SE}}(B)$.

We can now bound the difference $P_2 - P_3$ as follows:

$$\begin{aligned} P_2 - P_3 &= \Pr_2[\,\mathbf{ExpH}_2 = 1\,] - \Pr_3[\,\mathbf{ExpH}_3 = 1\,] \\ &= \Pr_2\left[\,\mathbf{ExpH}_2 = 1 \wedge \overline{E_2}\,\right] + \Pr_2[\,\mathbf{ExpH}_2 = 1 \wedge E_2\,] \\ &\quad - \Pr_3\left[\,\mathbf{ExpH}_3 = 1 \wedge \overline{E_3}\,\right] - \Pr_3[\,\mathbf{ExpH}_3 = 1 \wedge E_3\,] \\ &\leq \mathbf{Adv}^{\text{priv-dcpa}}_{\mathcal{SE}}(B) + 2 \cdot \mathbf{Adv}^{\text{coll-cpa}}_{\mathcal{EC}}(C)\;. \end{aligned}$$

To justify Claim 3.7.8, let E_0 be the event that there exists at least one collision among the $M_{t,0}$'s in $\mathbf{ExpH}_2$ and let E_1 be the event that there exists at least one collision among the $M_{t,1}$'s in $\mathbf{ExpH}_2$. Let $\Pr[\,\cdot\,]$ be over $\mathbf{Exp}^{\text{coll-cpa}}_{\mathcal{EC}}(C)$. Then,

$$\begin{aligned} \Pr\left[\,\mathbf{Exp}^{\text{coll-cpa}}_{\mathcal{EC}}(C) = 1\,\right] &= \Pr[\,E_0 \wedge d = 0\,] + \Pr[\,E_1 \wedge d = 1\,] \\ &= \frac{1}{2} \cdot \Big(\Pr_2[\,E_0\,] + \Pr_2[\,E_1\,]\Big) \geq \frac{1}{2} \cdot \Pr_2[\,E_2\,] \end{aligned}$$

where the second equality comes from the fact that the messages C returns to A are independent of the bit d. To justify Claim 3.7.9, we note that B returns 1 only if all the $M_{t,0}$'s and $M_{t,1}$'s are unique (i.e., events E_2 or E_3 did not occur).

The adversaries B and C use the same resources as S except that the queries that B makes to its oracle may be slightly larger than those of S due to the encoding. Also, B and C each perform two encodings for each query that S makes and, thus, their running times are (polynomially) larger than that of S. Recall the standard convention that running time of an adversary is measured with respect to the entire experiment in which it runs. Hence, Lemma 3.7.7 follows. ∎

3.8 SSH Security Results

Figure 3.2 shows the SSH encoding scheme when used with an l-bit block cipher; see also Section 3.2. Recall that $|x|$ denotes the length of string x in bits, not octets, and that $\langle x \rangle_k$ denotes the representation of x as a k-bit unsigned integer. The states st_n and st_u are maintained across invocations. As mentioned, although Figure 3.2 shows the padding p as a random string (the second boxed equation), the SSH specification does not require that p be random. Additionally, although the SSH specification allows up to 255 octets of padding, the two major implementations of the SSH protocol that we evaluated, `openssh-2.9p2` and SSH Communications' `ssh-3.0.1`, use the minimum-recommended padding length shown in Figure 3.2. The proposed SSH-$NPC instantiation of SSH replaces the first boxed statement with $\mathsf{bpl} \leftarrow \mathsf{bpl} + l$ if $\mathsf{bpl} < l$ and *always* uses random padding as shown in the second boxed statement. The instantiations SSH-CTRIV-CBC, SSH-EIV-CBC, and SSH-CTR, on the other hand, uses the first boxed statement with no modification and allows padding p to be non-random.

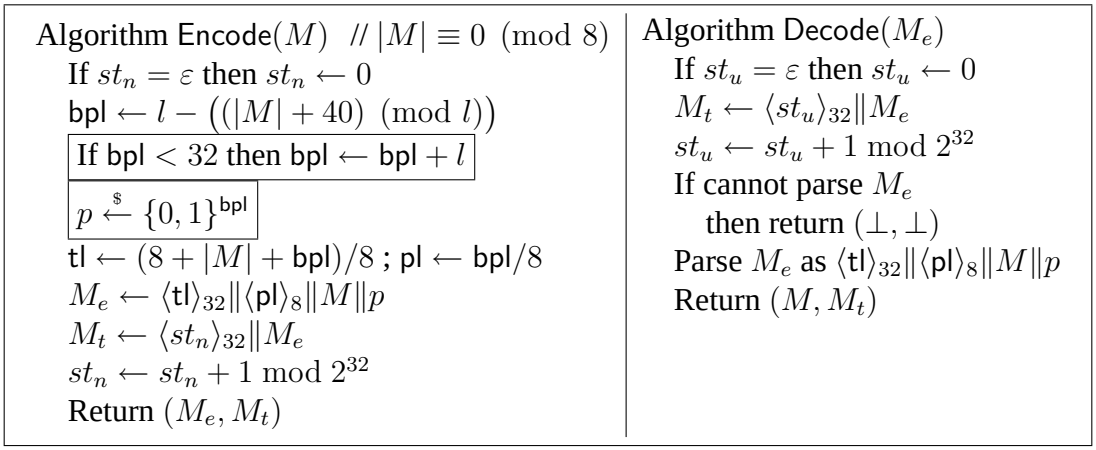

Figure 3.2 The SSH encoding scheme $\mathcal{EC} = (\mathsf{Encode}, \mathsf{Decode})$ for l-bit blocks, where $l \equiv 0 \pmod 8$ and $64 \leq l \leq 252 \cdot 8$.

3.8.1 Collision-Resistance of the SSH Encoding Scheme

The following lemma gives the collision bounds for the SSH encoding as shown in Figure 3.2. Notice that if $q_e \leq 2^{32}$, then $\lceil q_e \cdot 2^{-32} \rceil - 1 \leq 0$ and $\mathbf{Adv}^{\text{coll-cpa}}_{\mathcal{EC}}(A) = 0$ for any adversary A. Also, if a COLL-CCA adversary C submits more than 2^{32} encoding queries or 2^{32} decoding queries, then it can completely break the scheme, i.e. $\mathbf{Adv}^{\text{coll-cca}}_{\mathcal{EC}}(C) = 1$. For COLL-CCA security of up to 2^{32} decoding queries it is critical that the decoding algorithm increment its counter on every invocation, even for messages that do not correctly decode.

Lemma 3.8.1 (Collision resistance of the SSH encoding.) Let $\mathcal{EC}$ be the encoding scheme shown in Figure 3.2 and let mbpl be the minimum padding length (32 bits in Figure 3.2; the 32 in the equations below corresponds to the length of the encoding scheme's internal counter, not the minimum padding length). For any COLL-CPA adversary A and any COLL-CCA adversary B, each making q_e encoding queries and, in the case of B, making q_d decoding queries, we have that

$$\begin{aligned} \mathbf{Adv}^{\text{coll-cpa}}_{\mathcal{EC}}(A) &\leq \lceil q_e \cdot 2^{-32} \rceil \cdot \left(\lceil q_e \cdot 2^{-32} \rceil - 1 \right) \cdot 2^{31 - \mathsf{mbpl}} \\ \mathbf{Adv}^{\text{coll-cca}}_{\mathcal{EC}}(B) &= 0 \quad \text{if } q_e, q_d \leq 2^{32} \end{aligned}$$

and that COLL-CCA collision resistance is not provided if q_e or $q_d > 2^{32}$. ∎

Proof of of Lemma 3.8.1: First, we prove the inequality in the theorem. Recall that the padding is chosen independently at random from $\{0,1\}^{\mathsf{mbpl}}$ where mbpl is the minimum padding length. For a COLL-CPA adversary A to win, it must make at least two encoding queries M^i, M^j such that $i \neq j$ and $M_t^i = M_t^j$. From the construction, this means that the values of the counters and the paddings must collide (i.e. $st_n^i = st_n^j$ and $p^i = p^j$). For each counter value, the probability that the paddings collide is $2^{-\mathsf{mbpl}}$. There are 2^{32} possible values for the counter, and each value occurs at most $\lceil q_e/2^{32} \rceil$ times over the course of the experiment. Therefore, the probability that any coll-cpa adversary A making at most q_e encoding queries can win is at most

$$\binom{\lceil q_e \cdot 2^{-32} \rceil}{2} \cdot 2^{32} \cdot 2^{-\mathsf{mbpl}}$$

After some simplification, the first inequality in the theorem follows.

Now, we prove the equality in the theorem. Recall that the construction in Figure 3.2 specifies that $M_t \leftarrow \langle st_n \rangle_{32} \| M_e$ for the encoding and that $M_t \leftarrow \langle st_u \rangle_{32} \| M_e$ for the decoding. Since the states st_n, st_u are counters that are maintained internally by the oracles, no adversary B can have control over them. Since both states start at 0, if B is limited to fewer than 2^{32} encoding queries and 2^{32} decoding queries, then it is easy to see that B cannot possibly make two queries satisfying either of the first two conditions in the experiment $\mathbf{Exp}_{\mathcal{EC}}^{\text{coll-cca}}(B)$. We now turn our attention to the last condition in the experiment and argue that B cannot possibly satisfy it either. Suppose toward a contradiction that B can somehow make a query M^i to $\mathsf{Encode}(\cdot)$ and a query m_e^j to $\mathsf{Decode}(\cdot)$ such that ($i \neq j$ or $M^i \neq m^j$ or $M_e^i \neq m_e^j$) and $M_t^i = m_t^j$ where $i, j \leq 2^{32}$. From Figure 3.2, $M_t^i = m_t^j$ implies that $M_e^i = m_e^j$ and consequently that $M^i = m^j$. Therefore, for this condition to be satisfied i must be different from j. However, $i, j \leq 2^{32}$. Therefore, $i \neq j$ implies that $st_n^i \neq st_u^j$. Therefore, $M_t^i \neq m_t^j$, and we have a contradiction. Thus, $\mathbf{Adv}_{\mathcal{EC}}^{\text{coll-cca}}(B) = 0$. ∎

3.8.2 Integrity and Privacy of Our Recommendations

We have already provided enough information (Theorems 3.7.1 and 3.7.2 and Lemma 3.8.1) to show that our fixes from Section 3.5 are secure under the notions of chosen-plaintext indistinguishability (PRIV-CPA) and integrity of plaintexts (AUTHP). But we can prove a much stronger result, namely, that our proposed fixes are secure under our strong notions of chosen-ciphertext indistinguishability (PRIV-SFCCA) and integrity of ciphertexts (AUTHSF). We present our proof of security for SSH-CTR. The proof technique extends naturally to other possible fixes to the SSH BPP.

Theorem 3.8.2 (Security of SSH-CTR.) Let $\mathcal{SE}$ be a CTR-mode encryption scheme with stateful decryption, let $\mathcal{MA}$ be a message authentication scheme, and let $\mathcal{EC}$ be the encoding scheme described above. Let SSH-CTR be the encryption scheme associated to them as per Construction 3.6.1. Then, given any AUTHSF adversary I against SSH-CTR, we can construct adversaries F and C such that Equation 3.3 holds. Similarly, given any PRIV-SFCCA adversary A against SSH-CTR, we can construct adversaries S, B, E, and G such that Equation 3.4 holds.

$$\mathbf{Adv}^{\text{authsf}}_{\text{SSH-CTR}}(I) \leq \mathbf{Adv}^{\text{uf}}_{\mathcal{MA}}(F) + \mathbf{Adv}^{\text{coll-cca}}_{\mathcal{EC}}(C) \tag{3.3}$$

$$\begin{aligned}\mathbf{Adv}^{\text{priv-sfcca}}_{\text{SSH-CTR}}(A) &\leq 2 \cdot \mathbf{Adv}^{\text{authsf}}_{\text{SSH-CTR}}(S) + \mathbf{Adv}^{\text{priv-cpa}}_{\mathcal{SE}}(B) \\ &\quad +2 \cdot \mathbf{Adv}^{\text{prf}}_{\mathcal{MA}}(E) + 2 \cdot \mathbf{Adv}^{\text{coll-cpa}}_{\mathcal{EC}}(G)\end{aligned} \tag{3.4}$$

Furthermore, F and C use the same resources as I except that F's messages to its oracles may be of different lengths than I's queries to its oracles (due to encoding) and C's messages to its decoding oracle may have slightly different lengths than I's decryption queries. Also, S, B, E, and G use the same resources as A except that B's and E's inputs to their respective oracles may be of different lengths than those of A (due to the encoding). ∎

We interpret Theorem 3.8.2 as follows. Equation 3.3 states that SSH-CTR provides stateful chosen-ciphertext integrity, AUTHSF-security, if the MAC is strongly unforgeable and if the encoding is COLL-CCA collision resistant. Equation 3.4 states that

SSH-CTR provides stateful chosen-ciphertext privacy, PRIV-SFCCA-security, if it provides AUTHSF stateful chosen-ciphertext integrity, if the underlying encryption scheme is PRIV-CPA-secure, if the MAC is a secure pseudorandom function, and if the encoding is COLL-CPA-secure. As an example, making reasonable assumptions about the security of the HMAC scheme, an implementation of SSH-CTR that uses HMAC and AES in stateful-decryption CTR mode will be secure under both of the strong notions provided that at most 2^{32} messages are encrypted between rekeying. Notice here that we use different security properties of the MAC to obtain different security aspects of SSH-CTR, namely strong unforgeability for integrity and pseudorandomness for privacy. This distills the property of the MAC that leads to each aspect of security. Now we present the proof of Theorem 3.8.2.

Proof of of Theorem 3.8.2: First we provide a proof sketch. To prove Theorem 3.8.2, we first use Lemma 3.8.1, Theorem 3.7.1, the PRIV-CPA proof of security for CTR mode [4], and the assumed pseudorandomness of the underlying MAC to show that SSH-CTR is PRIV-CPA-secure. We then prove Equation 3.3. Applying Proposition 3.6.4 and our PRIV-CPA and AUTHSF results for SSH-CTR leads to Equation 3.4. We briefly discuss our proof of Equation 3.3. Let I be an AUTHSF adversary and let M^i be I's i-th chosen-plaintext query to its encoding oracle, let M_e^i, M_t^i be the encoding of M^i, and let $\sigma_i \| \tau_i$ be the returned ciphertext. Let $\sigma'_j \| \tau'_j$ be I's j-th decryption-verification oracle query, let m_e^j be the decryption of σ'_j by the underlying decryption algorithm. To prove Equation 3.3, we show that given an AUTHSF adversary attacking SSH-CTR, that adversary can also be used to attack the unforgeability of the underlying MAC, to attack the COLL-CCA collision resistance of the underlying encoding scheme, or that the first out-of-order ciphertext submitted by the adversary, $\sigma'_j \| \tau'_j$, is such that $\sigma_j \neq \sigma'_j$ but $M_e^j = m_e^j$. By properties of CTR mode with stateful decryption, the latter event cannot occur. The same property holds for SSH-CTRIV-CBC and SSH-EIV-CBC. For SSH-$NPC the latter event can occur, but the probability the latter event occurs is small because the last (random) block of the encoded packet is not given to the adversary. The strategy we outlined in this paragraph can be used to prove the security of other fixes to

the SSH BPP that work by replacing the underlying encryption scheme; namely, prove that the underlying encryption scheme is PRIV-CPA-secure and that the probability of the event we described is small. (We only consider the first out-of-order ciphertext query an adversary makes because if the first out-of-order ciphertext query does not decrypt, the decryptor enters a halting state.)

Now we present the proof in more detail. First, we note that Equation 3.4 follows directly from Proposition 3.6.4 and Theorem 3.7.1. Now, we prove Equation 3.3. Let $\overline{\mathcal{SE}} = (\overline{\mathcal{K}}, \overline{\mathcal{E}}, \overline{\mathcal{D}})$ be the composite encryption scheme (SSH-CTR in this case) constructed via Construction 3.6.1 from the encryption scheme $\mathcal{SE} = (\mathcal{K}_e, \mathcal{E}, \mathcal{D})$, the MAC scheme $\mathcal{MA} = (\mathcal{K}_t, \mathcal{T}, \mathcal{V})$, and the encoding scheme $\mathcal{EC} = (\mathsf{Encode}, \mathsf{Decode})$. Consider any AUTHSF adversary I against $\overline{\mathcal{SE}}$. We associate to I a UF forger F against $\mathcal{MA}$ and a COLL-CCA collision finder C against $\mathcal{EC}$ as follows. The forger F uses $\mathcal{K}_e$ to generate an encryption key and uses the encryption key and its tagging oracle to answer I's queries in a straight-forward manner. In particular, it follows Construction 3.6.1. Similarly, the collision finder C uses the same approach. This ensures that I is executed in the same environment as that in $\mathbf{Exp}^{\text{authsf}}_{\overline{\mathcal{SE}}}(I)$ until I succeeds in making an out-of-sync query.

Recall that the AUTHSF adversary I can make two types of queries: encryption queries to $\overline{\mathcal{E}}_K$ and decryption-verification queries to $\overline{\mathcal{D}}^*_K$. Suppose I makes q_e encryption queries and q_d decryption-verification queries. We denote I's i-th query to $\overline{\mathcal{E}}_K$ as M^i, the encoding of M^i as M^i_e, M^i_t, and the returned ciphertext as $\sigma_i \| \tau_i$. We denote I's i-th query to $\overline{\mathcal{D}}^*_K$ as $\sigma'_i \| \tau'_i$ (assuming that I's i-th query is parsable since otherwise $\overline{\mathcal{D}}^*_K$ would enter a halting state). We denote the decryption (via $\mathcal{D}$) of σ'_i as m^i_e and the decoding of m^i_e as (m^i, m^i_t). By convention, if $\overline{\mathcal{D}}^*_k$'s internal state is $\perp$, then $m^i_e = \perp$. Also, if $m^i_e = \perp$, then $(m^i, m^i_t) = (\perp, \perp)$.

Now, let j be the index of I's first out-of-sync decryption query, and let k be the number of encryption queries prior to I's j-th decryption query. Let Bad be an event in which all of the following conditions hold: I's j-th decryption query correctly verifies, $m^j_t \in \{M^1_t, \ldots, M^k_t\}$, $k \geq j$, $\tau'_j = \tau_j$, and $m^j_e = M^j_e$. (Recall that if the first out-

of-sync decryption query fails to verify, the decryption algorithm will return $\perp$ for all subsequent decryption queries.) We state the following lemmas.

Lemma 3.8.3 $\mathbf{Adv}^{\text{authsf}}_{\overline{\mathcal{SE}}}(I) \leq \mathbf{Adv}^{\text{uf}}_{\mathcal{MA}}(F) + \mathbf{Adv}^{\text{coll-cca}}_{\mathcal{EC}}(C) + \Pr[\,\mathsf{Bad}\,]$

Lemma 3.8.4 $\Pr[\,\mathsf{Bad}\,] = 0$

Then, Equation 3.3 in Theorem 3.8.2 follows. ∎

Proof of of Lemma 3.8.3: Let $\Pr[\,\cdot\,]$ denote the probability function underlying the experiment $\mathbf{Exp}^{\text{authsf}}_{\overline{\mathcal{SE}}}(I)$. Let $\sigma'_j \| \tau'_j$ be I's first out-of-sync query to $\overline{\mathcal{D}}^*_K(\cdot)$. Recall that, prior to I's j-th decryption-verification query, I made k queries to $\overline{\mathcal{E}}_K(\cdot)$. We define the following events.

Event E	:	I's first out-of-sync query to oracle $\overline{\mathcal{D}}^*_K(\cdot)$ correctly verifies
Event E_1	:	E occurs and $m_t^j \notin \{M_t^1, \ldots, M_t^k\}$
Event E_2	:	E occurs and $m_t^j \in \{M_t^1, \ldots, M_t^k\}$
Event $E_{2,1}$	:	E_2 occurs and either $k < j$ or $m_e^j \neq M_e^j$
Event $E_{2,2}$	:	E_2 occurs and $k \geq j$ and $m_e^j = M_e^j$
Event $E_{2,2,1}$	:	$E_{2,2}$ occurs and $\tau'_j \neq \tau_j$ and $m_t^j \notin \{M_t^1, \ldots, M_t^{j-1}, M_t^{j+1}, \ldots, M_t^k\}$
Event $E_{2,2,2}$	:	$E_{2,2}$ occurs and $\tau'_j \neq \tau_j$ and $m_t^j \in \{M_t^1, \ldots, M_t^{j-1}, M_t^{j+1}, \ldots, M_t^k\}$
Event $E_{2,2,3}$	:	$E_{2,2}$ occurs and $\tau'_j = \tau_j$.

If I's first out-of-sync query to $\overline{\mathcal{D}}^*_K(\cdot)$ does not correctly verify, then the decryption oracle enters its halting state, and thus, no further decryption queries will correctly verify and $\mathbf{Exp}^{\text{authsf}}_{\overline{\mathcal{SE}}}(I)$ cannot return 1. Therefore, $\mathbf{Adv}^{\text{authsf}}_{\overline{\mathcal{SE}}}(I) = \Pr[\,E\,]$. Also, notice that $\Pr[\,E\,] = \Pr[\,E_1 \vee E_{2,2,1}\,] + \Pr[\,E_{2,1} \vee E_{2,2,2}\,] + \Pr[\,E_{2,2,3}\,]$.

As previously pointed out, the adversaries F and C run I exactly as in experiment $\mathbf{Exp}^{\text{authsf}}_{\overline{\mathcal{SE}}}(I)$ until I succeeds in making an out-of-sync decryption-verification query. Therefore, if events E_1 or $E_{2,2,1}$ occur, then F succeeds in finding a UF forgery against

$\mathcal{MA}$. Similarly, if events $E_{2,1}$ or $E_{2,2,2}$ occur, C succeeds in finding a collision against $\mathcal{EC}$. Consequently,

$$\begin{aligned}\mathbf{Adv}^{\text{authsf}}_{\mathcal{SE}}(I) &= \Pr[\, E_1 \vee E_{2,2,1} \,] + \Pr[\, E_{2,1} \vee E_{2,2,2} \,] + \Pr[\, E_{2,2,3} \,] \\ &\leq \mathbf{Adv}^{\text{uf}}_{\mathcal{MA}}(F) + \mathbf{Adv}^{\text{coll-cca}}_{\mathcal{EC}}(C) + \Pr[\, \mathsf{Bad} \,]\end{aligned}$$

as desired. ∎

Proof of of Lemma 3.8.4: We are interested in the event that $\sigma'_j \neq \sigma_j$ but $m^j_e = M^j_e$ (where j is the index of the first out-of-order decryption query and the adversary has already queried the encryption oracle at least j times). Since SSH-CTR uses CTR mode with stateful decryption, since the encryption and decryption states are in-sync prior to the j-th decryption query, and since, for each CTR mode state, there is a bijection between plaintexts and ciphertexts, if $\sigma'_j \neq \sigma_j$, then $m^j_e \neq M^j_e$. This means that $\Pr[\mathsf{Bad}] = 0$. ∎

3.9 Discussion and Recommendations

Having presented our main results, we are now in a position to make specific recommendations to the SSH community. We begin by noting that one problem with the current SSH specification is that the counter that is prepended to the encoded payload before MACing is only 32 bits long. As shown in Section 3.4, once the 32 bit counter repeats, an SSH session's MAC tags may begin to leak information about a user's plaintexts. Our provable security results reflect this constraint: strong security is maintained only if the parties rekey at least once every 2^{32} packets. Two natural solutions to this problem are to either make the counter longer or to require an SSH session to rekey at least once every 2^{32} messages. We recommend the second option because it does not affect the packet format and thus will likely require minimal changes to existing SSH implementations. As a slight variant of the first option, we do note that it would be possible to define new message authentication modules for SSH that maintain and update their own, longer counters; this approach would also not affect the packet format.

With respect to the underlying encryption mode, we now compare the current instantiation of the SSH BPP transport protocol, SSH-IPC, to our specific recommendations. We also consider two other possible alternatives, namely switching to an Encrypt-then-MAC-based construction or to a dedicated authenticated encryption construction. The former involves re-engineering the SSH BPP so that it first encrypts a message with some underlying encryption scheme and then MACs the resulting ciphertext. The latter involves modifying SSH to use a dedicated authenticated encryption scheme like XCBC [35] or OCB [72].

Continue to use SSH-IPC? As mentioned, SSH-IPC is susceptible to an adaptive chosen-plaintext attack requiring an SSH user to encrypt on the order of 2^{13} packets. However, the attack may not be considered practical since it requires the attacker to, after seeing a ciphertext collision, control the *next* message that a user encrypts. If the session is encrypting a lot of data very quickly (e.g., while transferring a file), then an attacker may not have time to both recognize that a collision has occurred and to force the user to encrypt a specially-doctored message, though an adversary might try to slow down the entire connection in anticipation of mounting a chosen-plaintext attack. Additionally, if we consider how the SSH transport protocol is used within SSH (and not as an entity by itself), then the attack is complicated by the fact that an application may compress and further encode user data before passing the resulting compressed payload to the SSH-IPC protocol. Nonetheless, we suggest that the use of SSH-IPC be deprecated. One simple reason is that, even if these attacks may be difficult to mount in practice, in the modern era of strong cryptography it would seem counterintuitive to voluntarily use a protocol with low security when it is possible to fix the security of SSH at low cost.

Switch to SSH-NPC? Since SSH-NPC requires similar changes to the specification and implementations as SSH-$NPC while achieving less security than our other fixes, there does not appear to be any substantial reasons to switch to SSH-NPC. Therefore, we do not consider it further.

**Switch to SSH-$NPC?** The advantages offered by SSH-$NPC are clear: it is provably secure and requires relatively minor and mostly localized changes to the SSH specification and to implementations. The added security, however, comes with the additional cost of up to two extra blocks per packet. In interactive sessions where an individual packet may only contain a few octets of user data, the additional cost associated with those extra blocks may be significant (in terms of bandwidth consumption, the time necessary to encrypt and MAC those two extra blocks, and the time required to generate the extra block of randomness). Another potential problem with SSH-$NPC is that it is prone to accidental implementation mistakes. Recall that if the padding used with SSH-$NPC is not randomized, then the same reaction attack against SSH-NPC will be effective here. Since two SSH implementations will inter-operate regardless of whether their padding is random or fixed, an SSH developer might accidentally use non-random or predictable padding. Such an accidental implementation mistake could have serious security consequences.

Switch to SSH-CTR? SSH-CTRIV-CBC? or SSH-EIV-CBC? The SSH-CTR instantiation is attractive since it is provably secure, does not incur packet expansion, and does not require the padding to be random. Furthermore, there are several performance advantages with using CTR mode instead of CBC mode; for example, a software CTR mode implementation can be up to four times faster than a well-optimized CBC implementation [55]. Although perhaps not as attractive as SSH-CTR, SSH-CTRIV-CBC and SSH-EIV-CBC are also promising candidates because they also require no additional padding and because they only use one more block cipher invocation per packet than SSH-IPC.

The underlying encryption schemes for the SSH-CTR, SSH-CTRIV-CBC, and SSH-EIV-CBC recommendations all require both the sender and the receiver to maintain state. Prior to this work, most provable security analyses focused on encryption schemes with stateless decryption algorithms (hence our need to define security notions for encryption schemes with stateful decryption algorithms). Consequently, one initial

objection to these three constructions might be that they require the underlying decryption algorithms to maintain state. However, since the composite SSH BPP decryption algorithm is already stateful (because the decoding algorithm is stateful), the fact that these three fixes use underlying encryption schemes with stateful decryption algorithms should be of little concern. Another potential disadvantage with CTR mode is that it is often perceived as being too "risky" [55]. As [55] points out, however, when used correctly and with proofs of security, CTR mode has many advantages over other encryption modes. Furthermore, as Bellovin and Blaze point out in [15], one can minimize the risk incurred with using CTR mode (including the risk of being forced to use repeating counters) if key management is done dynamically and properly, if it is not used with multiple senders who share keys, and if it is used in conjunction with strong integrity checks. All of these conditions hold in the case of SSH-CTR.

Switch to Encrypt-then-MAC? Instead of insisting on continuing to use the current SSH Encode-then-E&M construction, it would also be possible to switch to another paradigm such as Encrypt-then-MAC. This alternative is attractive because, as we recalled in Section 2.6, an Encrypt-then-MAC construction is provably secure assuming that its underlying encryption and message authentication schemes are also secure [10, 52]. We note, however, that since our recommended fixes provably meet our strongest notions of security, there may be little motivation to switch to an Encrypt-then-MAC-based construction. Additionally, switching to an Encrypt-then-MAC construction will likely require more intrusive modifications to the current SSH specification and to SSH implementations. Furthermore, unless care is taken, implementations of the modified SSH specification may not be compatible with implementations of the current SSH specification. Conceptually speaking, the changes incurred by SSH-CTR, SSH-$NPC, SSH-CTRIV-CBC, and SSH-EIV-CBC involve only changing the underlying encryption module and, in the case of SSH-$NPC, adding more random number generation for the padding. In contrast, the changes incurred by switching to the Encrypt-then-MAC construction involve changing the whole construction. (We ac-

knowledge that the difference in the actual efforts that developers need to exert to implement these changes will be highly implementation dependent.)

Switch to dedicated authenticated encryption schemes? As noted in Section 2.6.6, there are symmetric key-based authenticated encryption schemes that are designed from scratch and, thus, are potentially more efficient than schemes based on a black-box composition of off-the-shelf encryption and MAC components. Recall that currently the input to the SSH BPP's underlying encryption scheme is different from the input to the underlying MAC. There are two possible ways to incorporate a dedicated authenticated encryption scheme into SSH: (1) specifically re-design the SSH specification around a single authenticated encryption component or (2) somehow plug a dedicated authenticated encryption scheme into the current SSH design.

For option (1), as we mentioned when we considered the Encrypt-then-MAC paradigm, re-designing the SSH specification is probably not an attractive option. For option (2), the most logical way to incorporate a dedicated scheme into SSH would be to replace the current encryption scheme (CBC mode with chained IVs) with something like XCBC or OCB and to use the "none" message authentication scheme. As we argued for SSH-CTR, SSH-$NPC, SSH-CTRIV-CBC, and SSH-EIV-CBC, this modification should be fairly easy to do, and, given the efficiency of dedicated authenticated encryption schemes, could result in significant performance gains. The present drawback with this approach is that the current SSH specification does not include the 32-bit counter in the input to the underlying encryption scheme. Since, under this construction, the counter will not be bound to the input to the dedicated authenticated encryption scheme, this construction cannot protect against replay and out-of-order delivery attacks (while our proposed recommendations can). To rectify this situation, one would still have to modify more than just the "black-box" encryption component of the SSH BPP, perhaps by using an authenticated encryption with associated data scheme [69], which has the same drawbacks as possibility (1) above, or use as the underlying encryption scheme an authenticated encryption scheme with its own internal counter, which

we view as an inelegant, though still viable, solution.

Closing remarks. We acknowledge that there are many possible ways to fix the current problems with the SSH protocol. We are biased toward our recommended fixes, and in particular SSH-CTR, because they are "less intrusive" than the other possible modifications but are still efficient and secure.

Following the initial publication of our research results, we submitted an Internet-Draft to the IETF Secure Shell working group specifying our SSH-CTR recommendation. The working group and the IESG has since approved our specification as an RFC (standards-track document) [9] and our SSH-CTR recommendation is implemented in the latest version of the OpenSSH application.

Additional Information

An earlier version of the material in this chapter appears in the ACM Transactions on Information and System Security [8], copyright the ACM. I was a primary researcher for this work. The full citation for this work is:

> Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the Encode-then-Encrypt-and-MAC paradigm. *ACM Transactions on Information and System Security*, 7(2):206–241, May 2004.

4 Generalized Composition Methods for Authenticated Encryption

In this chapter we further extend our generalization of the generic Encrypt-and-MAC paradigm from Chapter 3, and we also formally study generalizations of the generic MAC-then-Encrypt and Encrypt-then-MAC paradigms. To differentiate the composition methods that we consider in this chapter from the earlier composition paradigms, we refer to the paradigms in this chapter as the *generalized Encode-then-E&M*, *Encode-then-MtE*, and *Encode-then-EtM* paradigms. See Figures 4.1–4.3.

We generalize our analysis in several ways. First, motivated by Rogaway [69] and properties of real protocols, we consider authenticated encryption schemes that can authenticate more data than they encrypt; note the two inputs to the encoding algorithms in Figures 4.1–4.3. Using Rogaway's terminology, we refer to such authenticated encryption schemes as *authenticated encryption with associated data* (AEAD) schemes.

We further broaden our study by considering five classes of AEAD schemes, all sharing the same syntax but having different consistency requirements and security goals. We refer to these five classes as Type 1 through Type 5 AEAD schemes. Intuitively, Type 1 AEAD schemes should accept any ciphertext, in any order, and possibly multiple times, as long as the ciphertext was generated by the encryptor. Type 1 AEAD schemes are equivalent to the AEAD schemes that Rogaway considers except that our Type 1 AEAD schemes do not take a nonce as input, we allow the encryption algorithms to be randomized and stateful, we allow the decryption algorithms to be stateful, and we

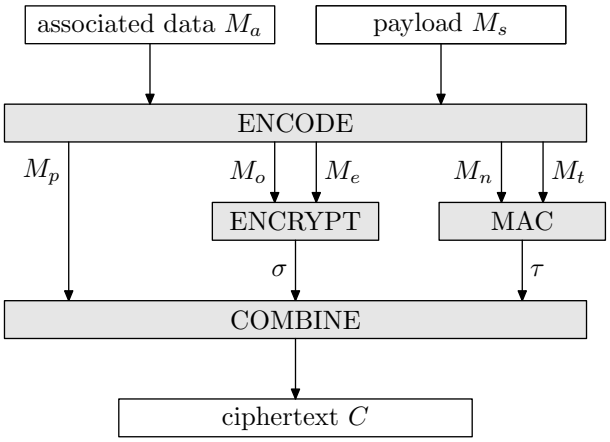

Figure 4.1 The generalized Encode-then-E&M paradigm.

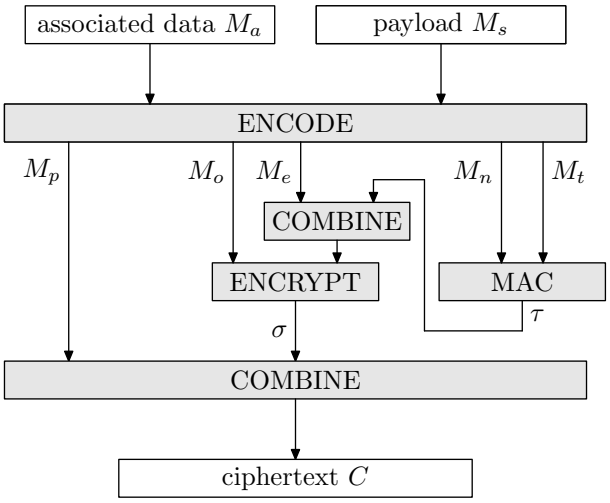

Figure 4.2 The generalized Encode-then-MtE paradigm.

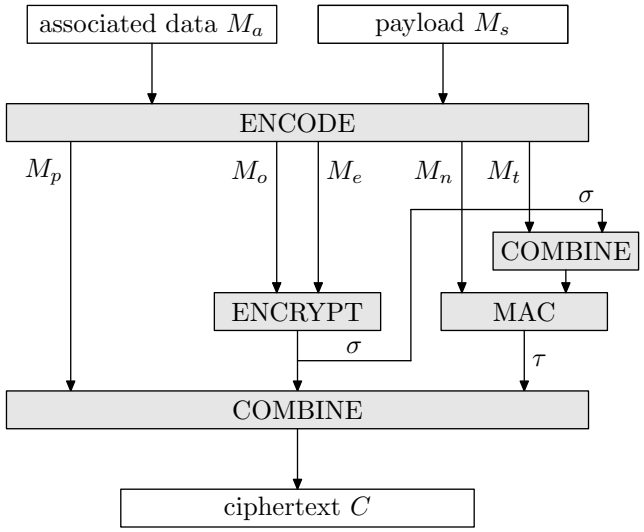

Figure 4.3 The generalized Encode-then-EtM paradigm.

base our definition of chosen-plaintext privacy on left-or-right indistinguishability; we do not expose a nonce to the caller because our goal is to model higher-level authenticated encryption schemes, and we believe that users of such authenticated encryption schemes should not be required to manipulate nonces. Type 2 AEAD schemes are like Type 1 AEAD schemes except that they should also protect against replay attacks. Type 3 AEAD schemes should only accept ciphertexts in monotonically increasing order of creation. Type 4 and Type 5 AEAD schemes are like Type 3 AEAD schemes except that they should only accept ciphertexts in exactly the order in which they were created. The latter two types differ in that a Type 4 AEAD scheme should halt after detecting a forgery attempt, and a Type 5 AEAD scheme should not. A Type 4 AEAD scheme is equivalent to the type of authenticated encryption schemes that we consider in Chapter 3, except that in Chapter 3 we do not handle associated data. As in Chapter 3, but unlike [10, 11, 47, 69], we allow the decryption algorithms for all five types of AEAD schemes to be stateful.

We also consider generalizations of the underlying encryption and message authentication components, and in particular we allow the encoding scheme to control the underlying encryption scheme's and MAC's initialization vectors (the M_o and M_n

variables in Figures 4.1–4.3). To contrast this approach with previous approaches, recall that it is traditional to assume that an underlying encryption scheme and MAC will internally handle the selection of any initialization vectors. Exposing the initialization vectors to the encoding schemes allow us to, for example, share initialization vectors between the encryption scheme and the MAC and to reuse the initialization vectors for other purposes, like protecting against replay attacks. When studying the privacy of the generalized Encode-then-E&M construction, we also consider the possibility of employing a privacy-preserving (PRIV-CPA-secure) MAC, such as UMAC [22]; recall that in Chapter 3 we focused our results on MACs that are PRF- or PRIV-DCPA-secure, not PRIV-CPA-secure. Considering PRIV-CPA-secure MACs allows us to prove the PRIV-CPA-security of a generalized Encode-then-E&M construction without placing requirements on the underlying encoding scheme, which we were not able to do in Chapter 3.

In Chapter 3 we proved PRIV-CPA and AUTHP results for the (non-generalized) Encode-then-E&M paradigm (Section 3.7), but restricted our chosen-ciphertext privacy (PRIV-CCA) and integrity of ciphertext (AUTHSF) results to SSH-CTR (Section 3.8). We address this deficiency here by presenting strong integrity of ciphertexts results for all three generalized composition paradigms and all five classes of AEAD schemes. Toward giving these results, we first introduce new properties for generalized Encode-then-E&M and generalized Encode-then-MtE constructions. Informally, the purpose of these properties are to capture the probability of events like Bad in the proof of security for SSH-CTR (recall Lemmas 3.8.3 and 3.8.4). We call these new properties E&M-SP and MTE-SP and note that there are common scenarios for which it is easy to determine if a generalized Encode-then-E&M or generalized Encode-then-MtE construction satisfies the respective property (Propositions 4.5.3 and 4.6.3).

4.1 Authenticated Encryption with Associated Data

We begin by formally defining what we mean by an AEAD scheme. Recall that our definition of a Type 1 AEAD scheme is identical to Rogaway's definition of an

AEAD scheme [69] except that we do not expose a nonce to the caller, we allow the encryption algorithm to be randomized and stateful, we allow the decryption algorithm to be stateful, and we use a notion of chosen-plaintext privacy based on left-or-right indistinguishability. Further recall that a Type 5 AEAD scheme is identical to the type of authenticated encryption scheme that we considered in Chapter 3 except that here we allow the encryption algorithm to accept associated data as input.

4.1.1 Syntax

A Type n, $n \in \{1, \ldots, 5\}$, *authenticated encryption with associated data (AEAD) scheme* $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ consists of three algorithms and is defined for some key space $\mathsf{KeySp}_{\mathcal{AE}}$, associated-data space $\mathsf{AdSp}_{\mathcal{AE}}$, and message space $\mathsf{MsgSp}_{\mathcal{AE}}$. The randomized key-generation algorithm $\mathcal{K}$ returns a key $K \in \mathsf{KeySp}_{\mathcal{AE}}$; we write $K \xleftarrow{\$} \mathcal{K}$. The possibly randomized and possibly stateful encryption algorithm $\mathcal{E}$ takes a key $K \in \mathsf{KeySp}_{\mathcal{AE}}$, associated data $M_a \in \mathsf{AdSp}_{\mathcal{AE}}$, and a message $M_s \in \mathsf{MsgSp}_{\mathcal{AE}}$, and outputs a ciphertext $C \in \{0,1\}^*$; we write $C \xleftarrow{\$} \mathcal{E}_K(M_a, M_s)$. The deterministic and possibly stateful decryption algorithm $\mathcal{D}$ takes a key $K \in \mathsf{KeySp}_{\mathcal{AE}}$ and a message $C \in \{0,1\}^*$, and outputs a pair of messages $(M_a, M_s) \in \mathsf{AdSp}_{\mathcal{AE}} \times \mathsf{MsgSp}_{\mathcal{AE}}$ or the pair $(\perp, \perp)$; we write $(M_a, M_s) \leftarrow \mathcal{D}_K(C)$. We say that $\mathcal{D}_K$ *accepts* C if $\mathcal{D}_K(C) \neq (\perp, \perp)$; otherwise $\mathcal{D}_K$ *rejects* C.

4.1.2 Consistency and Security

As is tradition and as we do elsewhere in this dissertation, we distinguish between the consistency requirements for AEAD schemes and their security goals. We first state a chosen-plaintext privacy goal, which is identical for the five types of AEAD schemes. We then consider each AEAD type individually, presenting first its integrity goal and then its consistency requirements. We state the security goals first since, in some cases, the consistency requirements need only be met if an adversary has not already succeeded in breaching the security of the scheme; for example, if an adversary forges a message,

it may place the decryptor in a state that it cannot recover from. We then restate the security properties in pseudocode for clarity.

Chosen-plaintext privacy. Our notion of privacy for AEAD schemes is similar to Rogaway's notion [69], which we will later use in Section 5, but here we use a formalization based on left-or-right-indistinguishability [4] and we do not expose a nonce to the adversary. Let $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an AEAD scheme. For $K \in \mathsf{KeySp}_{\mathcal{AE}}$ and $b \in \{0,1\}$, let $\mathcal{E}_K(\cdot, \mathcal{LR}(\cdot,\cdot,b))$ denote a *left-or-right (LR) encryption oracle* that takes input $M_a \in \mathsf{AdSp}_{\mathcal{AE}}$ and $M_0, M_1 \in \mathsf{MsgSp}_{\mathcal{AE}}$, and returns $\mathcal{E}_K(M_a, M_b)$. Let A be an adversary that returns a bit and has access to an LR encryption oracle. We require that for each query M_a, M_0, M_1 that A makes, $|M_0| = |M_1|$. We define the PRIV-CPA-*advantage* of PRIV-CPA-*adversary* A as

$$\begin{aligned}\mathbf{Adv}^{\text{priv-cpa}}_{\mathcal{AE}}(A) \quad &= \quad \Pr\left[\, K \xleftarrow{\$} \mathcal{K} \;:\; A^{\mathcal{E}_K(\cdot,\mathcal{LR}(\cdot,\cdot,1))} = 1 \,\right] \\ &\quad\; - \Pr\left[\, K \xleftarrow{\$} \mathcal{K} \;:\; A^{\mathcal{E}_K(\cdot,\mathcal{LR}(\cdot,\cdot,0))} = 1 \,\right] .\end{aligned}$$

In the concrete setting [6], we say that $\mathcal{AE}$ preserves privacy under chosen-plaintext attacks (or is PRIV-CPA *secure*) if the PRIV-CPA-advantage of all PRIV-CPA-adversaries using reasonable resources is small.

Integrity and chosen-ciphertext privacy. For $n \in \{1, \ldots, 5\}$, the integrity notion for a Type n AEAD scheme, AUTHn, addresses the authenticity of the *ciphertexts* generated by the encryption algorithm. As with classic authenticated encryption schemes (recall Section 2.6 and [10]), this is different from protecting the integrity of the original inputs to the encryption method. Indeed, the latter, in combination with the PRIV-CPA notion, is insufficient to guarantee privacy under chosen-ciphertext attacks, whereas AUTHn security together with PRIV-CPA security imply a strong notion of privacy under chosen-ciphertext attacks that we call PRIV-CCAn security. Since PRIV-CPA and AUTHn imply PRIV-CCAn, we focus all our discussions on the former two notions, but present the PRIV-CCAn notions later for completeness.

Throughout the following discussions, let $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a Type n, $n \in \{1, \ldots, 5\}$, AEAD scheme, the exact type will be clear from context. For each authenticity notion, we consider an adversary A with access to an encryption oracle $\mathcal{E}_K(\cdot, \cdot)$ and a decryption-verification oracle $\mathcal{D}_K^*(\cdot)$; the latter, on input C, invokes $\mathcal{D}_K(C)$ and returns 1 (i.e., accepts) if $\mathcal{D}_K(C) \neq (\perp, \perp)$ and 0 (i.e., rejects) otherwise. We define the AUTHn*-advantage* of AUTHn*-adversary* A, $\mathbf{Adv}^{\text{auth}n}_{\mathcal{AE}}(A)$, as the probability that A "forges" when given access to $\mathcal{E}_K(\cdot, \cdot)$ and $\mathcal{D}_K^*(\cdot)$ for a randomly selected K. We describe what "forges" means below. In the concrete setting [6], we say that a Type n AEAD scheme preserves authenticity if the AUTHn-advantage of all AUTHn-adversaries using reasonable resources is small.

Type 1. We say that an AUTH1-adversary "forges" if it makes the $\mathcal{D}_K^*(\cdot)$ oracle accept a ciphertext not previously returned by $\mathcal{E}_K(\cdot, \cdot)$. This security definition is similar to the AUTHC [10, 11, 47] definition recalled in Section 2.6 and Rogaway's integrity definition [69] except that here we consider AEAD schemes, not basic authenticated encryption schemes, and our AEAD schemes do not take nonces as input. The consistency requirement for Type 1 AEAD schemes is that $\mathcal{D}_K(\mathcal{E}_K(M_a, M_s)) = (M_a, M_s)$ for all $M_a \in \mathsf{AdSp}_{\mathcal{AE}}$, $M_s \in \mathsf{MsgSp}_{\mathcal{AE}}$, $K \in \mathsf{KeySp}_{\mathcal{AE}}$, and all internal states and random tapes of the encryptor and decryptor.

Type 2. We say that an AUTH2-adversary "forges" if it makes the $\mathcal{D}_K^*(\cdot)$ oracle accept a ciphertext not previously returned by $\mathcal{E}_K(\cdot, \cdot)$, or makes it accept the same ciphertext twice. For consistency, we require that for all $M_a \in \mathsf{AdSp}_{\mathcal{AE}}$, $M_s \in \mathsf{MsgSp}_{\mathcal{AE}}$ and $K \in \mathsf{KeySp}_{\mathcal{AE}}$, if $C = \mathcal{E}_K(M_a, M_s)$ for any internal state and random tape of the encryptor, C has not been submitted to $\mathcal{D}_K$, and an adversary has not succeeded in forging, then $\mathcal{D}_K(C) = (M_a, M_s)$. We also require that for any two message pairs (M_a^1, M_s^1), (M_a^2, M_s^2), if the encryptor computes $C_1 \xleftarrow{\$} \mathcal{E}_K(M_a^1, M_s^1)$ at some point in time and $C_2 \xleftarrow{\$} \mathcal{E}_K(M_a^2, M_s^2)$ at some other time, it is the case that $C_1 \neq C_2$ (even if $(M_a^1, M_s^1) = (M_a^2, M_s^2)$). Otherwise, a legitimately encrypted message might incorrectly be rejected by the receiver.

Type 3. For Type 3 AEAD schemes, we say that an AUTH3-adversary "forges" if it can make the $\mathcal{D}_K^*(\cdot)$ oracle accept a ciphertext not previously returned by $\mathcal{E}_K(\cdot,\cdot)$, accept the same ciphertext twice, or accept a ciphertext that was returned by $\mathcal{E}_K(\cdot,\cdot)$ before the last accepted ciphertext. For consistency, we require that for all $M_a \in \mathsf{AdSp}_{\mathcal{AE}}$, $M_s \in \mathsf{MsgSp}_{\mathcal{AE}}$ and $K \in \mathsf{KeySp}_{\mathcal{AE}}$, if $C = \mathcal{E}_K(M_a, M_s)$ for any internal state and random tape of the encryptor, C or a ciphertext generated after C has not been submitted to $\mathcal{D}_K$, and an adversary has not succeeded in forging, then $\mathcal{D}_K(C) = (M_a, M_s)$. We also require that for any two message pairs (M_a^1, M_s^1), (M_a^2, M_s^2), if the encryptor computes $C_1 \xleftarrow{\$} \mathcal{E}_K(M_a^1, M_s^1)$ at some point in time and $C_2 \xleftarrow{\$} \mathcal{E}_K(M_a^2, M_s^2)$ at some other point in time, it is the case that $C_1 \neq C_2$ (even if $(M_a^1, M_s^1) = (M_a^2, M_s^2)$).

Type 4. The authenticity game for Type 4 AEAD scheme begins with a flag phase set to 0. If at any point the sequence of queries to the $\mathcal{D}_K^*(\cdot)$ oracle fails to be a prefix of the responses from $\mathcal{E}_K(\cdot,\cdot)$, phase is set to 1. The AUTH4-adversary "forges" if it can force the $\mathcal{D}_K^*(\cdot)$ oracle to accept a message after phase becomes 1. This security definition is similar to the notion of integrity that we introduced in Chapter 3 for SSH except that here we are considering AEAD schemes. Consider some sequence of message pairs $(M_a^1, M_s^1), (M_a^2, M_s^2), \ldots$ and, for $i = 1, 2, \ldots$, let $C_i = \mathcal{E}_K(M_a^i, M_s^i)$, starting with $\mathcal{E}_K$ in its initial state. Then, for consistency purposes, if $\mathcal{D}_K$ is run on the sequence $C_1, C_2, \ldots$ in order and without the injection of additional packets, we require that $\mathcal{D}_K(C_i) = (M_a^i, M_s^i)$.

Type 5. Let C_i denote the i-th ciphertext produced by $\mathcal{E}_K(\cdot,\cdot)$. An AUTH5-adversary "forges" if the first message the $\mathcal{D}_K^*(\cdot)$ oracle accepts is not C_1. Inductively, if the last ciphertext accepted by $\mathcal{D}_K^*(\cdot)$ was C_j, then an AUTH5-adversary "forges" if it can make the oracle accept any $C \neq C_{j+1}$ before being invoked with C_{j+1}. Let (M_a^1, M_s^1), $(M_a^2, M_s^2), \ldots$ denote a sequence of message pairs and $C_1, C_2, \ldots$ denote their encryption under $\mathcal{E}$ and any key K. For consistency, we require that if $\mathcal{D}_K$ has not yet accepted any message (i.e., $\mathcal{D}_K$ is in its initial state or has always returned $(\perp, \perp)$), then $\mathcal{D}_K(C_1) = (M_a^1, M_s^1)$. For $i \geq 1$, if the only packets accepted by $\mathcal{D}_K$ are $C_1, C_2, \ldots, C_i$,

in that order but with possibly some rejected packets in the sequence of messages given to $\mathcal{D}_K$, then $\mathcal{D}_K(C_{i+1}) = (M_a^{i+1}, M_s^{i+1})$.

Pseudocode for security definitions. For clarity, we now present in pseudocode our PRIV-CPA and AUTHn notions of security for AEAD schemes. For completeness, we also present out PRIV-CCAn chosen-ciphertext privacy notions. We also give Theorem 4.1.3, which shows that if a Type n AEAD schemes is both PRIV-CPA-secure and AUTHn-secure, then it is also PRIV-CCAn-secure. Like the traditional PRIV-CCA chosen-ciphertext privacy notion for encryption schemes recalled in Section 2.4 [4], our PRIV-CCAn notions give an adversary as much power as possible without allowing the adversary to trivially win the associated game. For example, in the experiment $\mathbf{Exp}^{\text{priv-cca1-b}}_{\mathcal{AE}}(A_1)$, the adversary A_1 against $\mathcal{AE}$, when querying its oracle $\mathcal{D}_K$ with input C, will only be returned (M_a, M_s) if $C \notin S$ (we present this definition in a slightly different way than we presented the PRIV-CCA definition in Section 2.4 since here we do not disallow the adversary from making decryption queries for $C \in S$, but rather have the experiment handle such a situation as a special case).

Definition 4.1.1 (Privacy.) Let $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an AEAD scheme and let $b \in \{0, 1\}$ be a bit. Let A_{cpa} be an adversary with access to an LR encryption oracle $\mathcal{E}_K(\cdot, \mathcal{LR}(\cdot, \cdot, b))$ and let A_1, A_2, A_3, A_5, and A_4 be adversaries with access to an LR encryption oracle and a decryption oracle $\mathcal{D}_K(\cdot)$. The behaviors of $\mathcal{E}_K(\cdot, \mathcal{LR}(\cdot, \cdot, b))$ and $\mathcal{D}_K(\cdot)$ are specified in the following experiments. Assume each adversary returns a bit. Consider the following experiments.

Experiment $\mathbf{Exp}^{\text{priv-cpa-b}}_{\mathcal{AE}}(A_{\text{cpa}})$
  $K \xleftarrow{\$} \mathcal{K}$
  Run $A_{\text{cpa}}^{\mathcal{E}_K(\cdot, \mathcal{LR}(\cdot, \cdot, b))}$
    Reply to $\mathcal{E}_K(M_a, \mathcal{LR}(M_0, M_1, b))$ queries as follows:
      $C \xleftarrow{\$} \mathcal{E}_K(M_a, M_b)$; $A_{\text{cpa}} \Leftarrow C$
  Until A_{cpa} returns a bit d
  Return d

Experiment $\mathbf{Exp}^{\text{priv-cca1-b}}_{\mathcal{AE}}(A_1)$

$K \xleftarrow{\$} \mathcal{K}$; $S \leftarrow \emptyset$

Run $A_1^{\mathcal{E}_K(\cdot,\mathcal{LR}(\cdot,\cdot,b)),\mathcal{D}_K(\cdot)}$

Reply to $\mathcal{E}_K(M_a, \mathcal{LR}(M_0, M_1, b))$ queries as follows:

$C \xleftarrow{\$} \mathcal{E}_K(M_a, M_b)$; $S \leftarrow S \cup \{C\}$; $A_1 \Leftarrow C$

Reply to $\mathcal{D}_K(C)$ queries as follows:

$(M_a, M_s) \leftarrow \mathcal{D}_K(C)$

If $C \notin S$ then $A_1 \Leftarrow (M_a, M_s)$

Until A_1 returns a bit d

Return d

Experiment $\mathbf{Exp}^{\text{priv-cca2-b}}_{\mathcal{AE}}(A_2)$

$K \xleftarrow{\$} \mathcal{K}$; $S \leftarrow \emptyset$; $S' \leftarrow \emptyset$

Run $A_2^{\mathcal{E}_K(\cdot,\mathcal{LR}(\cdot,\cdot,b)),\mathcal{D}_K(\cdot)}$

Reply to $\mathcal{E}_K(M_a, \mathcal{LR}(M_0, M_1, b))$ queries as follows:

$C \xleftarrow{\$} \mathcal{E}_K(M_a, M_b)$; $S \leftarrow S \cup \{C\}$; $A_2 \Leftarrow C$

Reply to $\mathcal{D}_K(C)$ queries as follows:

$(M_a, M_s) \leftarrow \mathcal{D}_K(C)$

If $C \notin S$ or $C \in S'$ then $A_2 \Leftarrow (M_a, M_s)$

If $(M_a, M_s) \neq (\perp, \perp)$ then $S' \leftarrow S' \cup \{C\}$

Until A_2 returns a bit d

Return d

Experiment $\mathbf{Exp}^{\text{priv-cca3-b}}_{\mathcal{AE}}(A_3)$

$K \xleftarrow{\$} \mathcal{K}$; $i \leftarrow 0$; $j \leftarrow 0$

Run $A_3^{\mathcal{E}_K(\cdot,\mathcal{LR}(\cdot,\cdot,b)),\mathcal{D}_K(\cdot)}$

Reply to $\mathcal{E}_K(M_a, \mathcal{LR}(M_0, M_1, b))$ queries as follows:

$i \leftarrow i + 1$; $C_i \xleftarrow{\$} \mathcal{E}_K(M_a, M_b)$; $A_3 \Leftarrow C_i$

Reply to $\mathcal{D}_K(C)$ queries as follows:

$(M_a, M_s) \leftarrow \mathcal{D}_K(C)$

If $C \notin \{C_{j+1}, \ldots, C_i\}$ then $A_3 \Leftarrow (M_a, M_s)$

Else $j \leftarrow$ index of C in $\{C_{j+1}, \ldots, C_i\}$

Until A_3 returns a bit d

Return d

Experiment $\mathbf{Exp}^{\text{priv-cca4-b}}_{\mathcal{AE}}(A_4)$

$K \xleftarrow{\$} \mathcal{K}$; $i \leftarrow 0$; $j \leftarrow 0$; $\mathsf{phase} \leftarrow 0$

Run $A_4^{\mathcal{E}_K(\cdot,\mathcal{LR}(\cdot,\cdot,b)),\mathcal{D}_K(\cdot)}$

Reply to $\mathcal{E}_K(M_a, \mathcal{LR}(M_0, M_1, b))$ queries as follows:

$i \leftarrow i + 1$; $C_i \xleftarrow{\$} \mathcal{E}_K(M_a, M_b)$; $A_4 \Leftarrow C_i$

Reply to $\mathcal{D}_K(C)$ queries as follows:

$j \leftarrow j + 1$; $(M_a, M_s) \leftarrow \mathcal{D}_K(C)$

If $j > i$ or $C \neq C_j$ then $\mathsf{phase} \leftarrow 1$

If $\mathsf{phase} = 1$ then $A_4 \Leftarrow (M_a, M_s)$

Until A_4 returns a bit d

Return d

Experiment $\mathbf{Exp}^{\text{priv-cca5-b}}_{\mathcal{AE}}(A_5)$

$K \xleftarrow{\$} \mathcal{K}$; $i \leftarrow 0$; $j \leftarrow 0$

Run $A_5^{\mathcal{E}_K(\cdot,\mathcal{LR}(\cdot,\cdot,b)),\mathcal{D}_K(\cdot)}$

Reply to $\mathcal{E}_K(M_a, \mathcal{LR}(M_0, M_1, b))$ queries as follows:

$i \leftarrow i + 1$; $C_i \xleftarrow{\$} \mathcal{E}_K(M_a, M_b)$; $A_5 \Leftarrow C_i$

Reply to $\mathcal{D}_K(C)$ queries as follows:

$(M_a, M_s) \leftarrow \mathcal{D}_K(C)$

If $j + 1 > i$ or $C \neq C_{j+1}$ then $A_5 \Leftarrow (M_a, M_s)$

If $(M_a, M_s) \neq (\perp, \perp)$ then $j \leftarrow j + 1$

Until A_5 returns a bit d

Return d

We require that for all queries M_a, M_0, M_1 to $\mathcal{E}_K(\cdot, \mathcal{LR}(\cdot, \cdot, b))$, $|M_0| = |M_1|$. We define the PRIV-CPA-*advantage* of PRIV-CPA-*adversary* A_{cpa}, and for $n = 1, \ldots, 5$, the

PRIV-CCAn-*advantage* of adversary A_n as

$$\begin{aligned}
\mathbf{Adv}^{\text{priv-cpa}}_{\mathcal{AE}}(A_{\text{cpa}}) &= \Pr\left[\,\mathbf{Exp}^{\text{priv-cpa-1}}_{\mathcal{AE}}(A_{\text{cpa}}) = 1\,\right] - \Pr\left[\,\mathbf{Exp}^{\text{priv-cpa-0}}_{\mathcal{AE}}(A_{\text{cpa}}) = 1\,\right] \\
\mathbf{Adv}^{\text{priv-cca}n}_{\mathcal{AE}}(A_n) &= \Pr\left[\,\mathbf{Exp}^{\text{priv-cca}n\text{-1}}_{\mathcal{AE}}(A_n) = 1\,\right] \\
&\quad - \Pr\left[\,\mathbf{Exp}^{\text{priv-cca}n\text{-0}}_{\mathcal{AE}}(A_n) = 1\,\right]. \quad \blacksquare
\end{aligned}$$

Definition 4.1.2 (Integrity.) Let $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an AEAD scheme. Let A_1, A_2, A_3, A_4, and A_5 be adversaries each with access to an encryption oracle $\mathcal{E}_K(\cdot,\cdot)$ and a decryption-verification oracle $\mathcal{D}^*_K(\cdot)$. The decryption-verification oracle, on input C, invokes $\mathcal{D}_K(C)$ and returns 1 if $\mathcal{D}_K(C) \neq (\perp, \perp)$ and 0 otherwise. Consider the experiments defined below. Each experiment returns 1 if the adversary "forges" and 0 otherwise.

Experiment $\mathbf{Exp}^{\text{auth1}}_{\mathcal{AE}}(A_1)$
  $K \xleftarrow{\$} \mathcal{K}$; $S \leftarrow \emptyset$
  Run $A_1^{\mathcal{E}_K(\cdot,\cdot),\mathcal{D}^*_K(\cdot)}$
    Reply to $\mathcal{E}_K(M_a, M_s)$ queries as follows:
      $C \xleftarrow{\$} \mathcal{E}_K(M_a, M_s)$; $S \leftarrow S \cup \{C\}$; $A_1 \Leftarrow C$
    Reply to $\mathcal{D}^*_K(C)$ queries as follows:
      $(M_a, M_s) \leftarrow \mathcal{D}_K(C)$
      If $(M_a, M_s) \neq (\perp, \perp)$ and $C \notin S$ then return 1
      If $(M_a, M_s) \neq (\perp, \perp)$ then $A_1 \Leftarrow 1$
      Else $A_1 \Leftarrow 0$
  Until A_1 halts
  Return 0

Experiment $\mathbf{Exp}^{\text{auth2}}_{\mathcal{AE}}(A_2)$
  $K \xleftarrow{\$} \mathcal{K}$; $S \leftarrow \emptyset$; $S' \leftarrow \emptyset$
  Run $A_2^{\mathcal{E}_K(\cdot,\cdot),\mathcal{D}^*_K(\cdot)}$
    Reply to $\mathcal{E}_K(M_a, M_s)$ queries as follows:
      $C \xleftarrow{\$} \mathcal{E}_K(M_a, M_s)$; $S \leftarrow S \cup \{C\}$; $A_2 \Leftarrow C$

Reply to $\mathcal{D}_K^*(C)$ queries as follows:

$(M_a, M_s) \leftarrow \mathcal{D}_K(C)$

If $(M_a, M_s) \neq (\perp, \perp)$ and ($C \notin S$ or $C \in S'$) then return 1

If $(M_a, M_s) \neq (\perp, \perp)$ then $S' \leftarrow S' \cup \{C\}$; $A_2 \Leftarrow 1$

Else $A_2 \Leftarrow 0$

Until A_2 halts

Return 0

Experiment $\mathbf{Exp}^{\text{auth3}}_{\mathcal{AE}}(A_3)$

$K \xleftarrow{\$} \mathcal{K}$; $i \leftarrow 0$; $j \leftarrow 0$

Run $A_3^{\mathcal{E}_K(\cdot,\cdot),\mathcal{D}_K^*(\cdot)}$

Reply to $\mathcal{E}_K(M_a, M_s)$ queries as follows:

$i \leftarrow i + 1$; $C_i \xleftarrow{\$} \mathcal{E}_K(M_a, M_s)$; $A_3 \Leftarrow C_i$

Reply to $\mathcal{D}_K^*(C)$ queries as follows:

$(M_a, M_s) \leftarrow \mathcal{D}_K(C)$

If $(M_a, M_s) \neq (\perp, \perp)$ and $C \notin \{C_{j+1}, \ldots, C_i\}$ then return 1

If $(M_a, M_s) \neq (\perp, \perp)$ then

$j \leftarrow$ index of C in $\{C_{j+1}, \ldots, C_i\}$; $A_3 \Leftarrow 1$

Else $A_3 \Leftarrow 0$

Until A_3 halts

Return 0

Experiment $\mathbf{Exp}^{\text{auth4}}_{\mathcal{AE}}(A_4)$

$K \xleftarrow{\$} \mathcal{K}$; $i \leftarrow 0$; $j \leftarrow 0$; $\mathsf{phase} \leftarrow 0$

Run $A_4^{\mathcal{E}_K(\cdot,\cdot),\mathcal{D}_K^*(\cdot)}$

Reply to $\mathcal{E}_K(M_a, M_s)$ queries as follows:

$i \leftarrow i + 1$; $C_i \xleftarrow{\$} \mathcal{E}_K(M_a, M_s)$; $A_4 \Leftarrow C_i$

Reply to $\mathcal{D}_K^*(C)$ queries as follows:

$j \leftarrow j + 1$; $(M_a, M_s) \leftarrow \mathcal{D}_K(C)$

If $j > i$ or $C \neq C_j$ then $\mathsf{phase} \leftarrow 1$

If $(M_a, M_s) \neq (\perp, \perp)$ and $\mathsf{phase} = 1$ then return 1

If $(M_a, M_s) \neq (\perp, \perp)$ then $A_4 \Leftarrow 1$

Else $A_4 \Leftarrow 0$

Until A_4 halts

Return 0

Experiment $\mathbf{Exp}^{\text{auth5}}_{\mathcal{AE}}(A_5)$

$K \xleftarrow{\$} \mathcal{K}$; $i \leftarrow 0$; $j \leftarrow 0$

Run $A_5^{\mathcal{E}_K(\cdot,\cdot),\mathcal{D}^*_K(\cdot)}$

Reply to $\mathcal{E}_K(M_a, M_s)$ queries as follows:

$i \leftarrow i + 1$; $C_i \xleftarrow{\$} \mathcal{E}_K(M_a, M_s)$; $A_5 \Leftarrow C_i$

Reply to $\mathcal{D}^*_K(C)$ queries as follows:

$(M_a, M_s) \leftarrow \mathcal{D}_K(C)$

If $(M_a, M_s) \neq (\perp, \perp)$ and $(j + 1 > i$ or $C \neq C_{j+1})$ then return 1

If $(M_a, M_s) \neq (\perp, \perp)$ then $j \leftarrow j + 1$; $A_5 \Leftarrow 1$

Else $A_5 \Leftarrow 0$

Until A_5 halts

Return 0

For $n = 1, \ldots, 5$, we define the AUTHn-*advantage* of AUTHn-*adversary* A_n as

$$\mathbf{Adv}^{\text{auth}n}_{\mathcal{AE}}(A_n) = \Pr\left[\, \mathbf{Exp}^{\text{auth}n}_{\mathcal{AE}}(A_n) = 1 \,\right]. \quad \blacksquare$$

Relations between notions. The following theorem shows that if a Type n AEAD scheme is both PRIV-CPA-secure and AUTHn-secure, then it is also PRIV-CCAn-secure. The proof of this theorem follows closely the structure of the proof of similar properties in [10, 47] and the proof of Proposition 3.6.4; we omit details.

Theorem 4.1.3 Let $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a Type n AEAD scheme. For any PRIV-CCAn adversary A, there exist a AUTHn adversary I and a PRIV-CPA adversary B such that

$$\mathbf{Adv}^{\text{priv-cca}n}_{\mathcal{AE}}(A) \;\leq\; 2 \cdot \mathbf{Adv}^{\text{auth}n}_{\mathcal{AE}}(I) + \mathbf{Adv}^{\text{priv-cpa}}_{\mathcal{AE}}(B)$$

and I and B use the same resources as A. $\blacksquare$

4.2 Building Blocks

4.2.1 Encryption Schemes

Syntax and consistency. We modify the definition of a symmetric encryption scheme from Section 2.4 to explicitly expose the IV to the caller, and possibly to the adversary. Our approach for exposing the IV to the caller is a generalization of exposing a nonce for encryption schemes [69, 71, 72]. Specifically, in this chapter a *symmetric encryption scheme* $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ consists of three algorithms and is defined for some key space $\mathsf{KeySp}_{\mathcal{SE}}$, IV space $\mathsf{IVSp}_{\mathcal{SE}}$, and message space $\mathsf{MsgSp}_{\mathcal{SE}}$. The randomized key-generation algorithm $\mathcal{K}$ returns a key $K \in \mathsf{KeySp}_{\mathcal{SE}}$; we write $K \xleftarrow{\$} \mathcal{K}$. The possibly randomized and stateful encryption algorithm $\mathcal{E}$ takes a key $K \in \mathsf{KeySp}_{\mathcal{SE}}$, an IV $I \in \mathsf{IVSp}_{\mathcal{SE}}$, and a message $M \in \mathsf{MsgSp}_{\mathcal{SE}}$, and returns a ciphertext $C \in \{0,1\}^*$; we write $C \xleftarrow{\$} \mathcal{E}_K^I(M)$. Example values for $\mathsf{IVSp}_{\mathcal{SE}}$ are $\{\varepsilon\}$ (when $\mathcal{SE}$ takes no IV) and $\{0,1\}^i$ for some positive integer i. The stateless and deterministic decryption algorithm $\mathcal{D}$ takes a key $K \in \mathsf{KeySp}_{\mathcal{SE}}$, an IV $I \in \mathsf{IVSp}_{\mathcal{SE}}$, and a ciphertext $C \in \{0,1\}^*$, and returns a message $M \in \{0,1\}^*$; we write $M \leftarrow \mathcal{D}_K^I(C)$. The following consistency requirement must be met: $\mathcal{D}_K^I(\mathcal{E}_K^I(M)) = M$ for all $M \in \mathsf{MsgSp}_{\mathcal{SE}}$, $I \in \mathsf{IVSp}_{\mathcal{SE}}$, $K \in \mathsf{KeySp}_{\mathcal{SE}}$, and any internal state and random tape of $\mathcal{E}$. Deviating from tradition, we consider three types of encryption schemes, which differ in their IV requirements: *nonced*, *length-based IV,* and *random IVed* encryption schemes, the details of which are below.

Privacy. Our notion of privacy for symmetric encryption schemes is based on the notion of left-or-right-indistinguishability under chosen-plaintext attacks [4] which we recall in Section 2.4. Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme. Let A be an adversary that returns a bit and has access to a *left-or-right (LR) encryption oracle* $\mathcal{E}_K(\cdot, \mathcal{LR}(\cdot,\cdot,b))$, for $K \in \mathsf{KeySp}_{\mathcal{SE}}$ and $b \in \{0,1\}$, which takes input $I \in \mathsf{IVSp}_{\mathcal{SE}}$ and M_0, $M_1 \in \mathsf{MsgSp}_{\mathcal{SE}}$, and returns $\mathcal{E}_K^I(M_b)$. We require that for each query I, M_0, M_1 that A makes, $|M_0| = |M_1|$. If $\mathcal{SE}$ is a *nonced encryption scheme,* we further require that all the IV's chosen by the adversary are unique. If $\mathcal{SE}$ is a *random IVed encryption*

scheme we require that the adversary always chooses the IV uniformly at random from $\mathsf{IVSp}_{\mathcal{SE}}$, and that the choice of IV is made *after* the plaintext pair M_0, M_1 to encrypt is determined. If $\mathcal{SE}$ is a *length-based IV* encryption scheme, we require that the first IV is randomly selected from $\mathsf{IVSp}_{\mathcal{SE}}$ as per the requirements for random IV encryption schemes, and that each subsequent IV is a deterministic function of the initial IV and the lengths of the previous plaintexts; we call this deterministic function the *length-based IV-deriving function* for the encryption scheme. (If $\mathsf{IVSp}_{\mathcal{SE}} = \{\varepsilon\}$, then the random IV is always ε, and this is how we model standard encryption schemes, which do not take IVs as input.) We define the PRIV-CPA-*advantage* of adversary A as

$$\begin{aligned}\mathbf{Adv}^{\text{priv-cpa}}_{\mathcal{SE}}(A) \quad &= \quad \Pr\left[\, K \xleftarrow{\$} \mathcal{K} \;:\; A^{\mathcal{E}_K(\cdot,\mathcal{LR}(\cdot,\cdot,1))} = 1 \,\right] \\ &\quad\; - \Pr\left[\, K \xleftarrow{\$} \mathcal{K} \;:\; A^{\mathcal{E}_K(\cdot,\mathcal{LR}(\cdot,\cdot,0))} = 1 \,\right] .\end{aligned}$$

In the concrete setting [6], we say that $\mathcal{SE}$ preserves privacy against adversaries that respect the IV properties of the scheme (or is PRIV-CPA *secure*) if the PRIV-CPA-advantage of all such adversaries using reasonable resources is small.

We note that in our constructions we can enforce the IV requirements through our use of *encodings*. Specifically, when attacking a generalized Encode-then-E&M, Encode-then-MtE, or Encode-then-EtM construction, an adversary will not have direct control over the inputs to the underlying encryption algorithm, and in particular may not be able to choose arbitrary IVs. Rather, the adversary against the composite AEAD scheme may only access the underlying encryption algorithm via the encoding algorithm.

4.2.2 Message Authentication Schemes

Syntax and consistency. As with symmetric encryption, in this chapter we consider message authentication schemes that expose an IV to the caller. Specifically, in this chapter a *message authentication scheme* $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ consists of three algorithms and is defined for some key space $\mathsf{KeySp}_{\mathcal{MA}}$, IV space $\mathsf{IVSp}_{\mathcal{MA}}$, message space $\mathsf{MsgSp}_{\mathcal{MA}}$, and tag space $\mathsf{TagSp}_{\mathcal{MA}}$. The randomized key-generation algorithm re-

turns a key $K \in \mathsf{KeySp}_{\mathcal{MA}}$; we write $K \xleftarrow{\$} \mathcal{K}$. The possibly randomized and stateful tagging algorithm takes a key $K \in \mathsf{KeySp}_{\mathcal{MA}}$, an IV $I \in \mathsf{IVSp}_{\mathcal{MA}}$, and a message $M \in \mathsf{MsgSp}_{\mathcal{MA}}$, and returns a tag $\tau \in \mathsf{TagSp}_{\mathcal{MA}}$; we write $\tau \xleftarrow{\$} \mathcal{T}_K^I(M)$. The deterministic and stateless verification algorithm takes a key $K \in \mathsf{KeySp}_{\mathcal{MA}}$, an IV $I \in \mathsf{IVSp}_{\mathcal{MA}}$, a message $M \in \mathsf{MsgSp}_{\mathcal{MA}}$, and a tag $\tau \in \{0,1\}^*$, and returns a bit; we write $b \leftarrow \mathcal{V}_K^I(M,\tau)$. The following consistency requirement must be met: $\mathcal{V}_K^I(M, \mathcal{T}_K^I(M)) = 1$ for all $M \in \mathsf{MsgSp}_{\mathcal{MA}}$, $I \in \mathsf{IVSp}_{\mathcal{MA}}$, $K \in \mathsf{KeySp}_{\mathcal{MA}}$, and any internal state and random tape of $\mathcal{T}$. We consider two types of MACs: *nonced* MACs and conventional MACs with $\mathsf{IVSp}_{\mathcal{MA}} = \{\varepsilon\}$; the latter are MACs that do not take nonces as input.

Unforgeability. The principle notion of security for MACs that we consider is based on strong unforgeability under chosen-message attacks [10]; recall Section 2.5. Let $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ be a MAC. Let F be an adversary with access to a tagging oracle $\mathcal{T}_K^{(\cdot)}(\cdot)$ and a verification oracle $\mathcal{V}_K^{(\cdot)}(\cdot,\cdot)$, for $K \in \mathsf{KeySp}_{\mathcal{MA}}$. If $\mathcal{MA}$ is a *nonced MAC*, we require that all the IV's chosen by the adversary when invoking the $\mathcal{T}_K^{(\cdot)}(\cdot)$ oracle are unique. We say that F "forges" if it makes a verification query I, M, τ such that $\mathcal{V}_K^I(M,\tau) = 1$ and τ was never returned by $\mathcal{T}_K^{(\cdot)}(\cdot)$ in response to query I, M. We define the UF-*advantage* of *forger* F as

$$\mathbf{Adv}^{\mathrm{uf}}_{\mathcal{MA}}(F) = \Pr\left[\, K \xleftarrow{\$} \mathcal{K} \;:\; F^{\mathcal{T}_K^{(\cdot)}(\cdot),\mathcal{V}_K^{(\cdot)}(\cdot,\cdot)} \text{ forges} \,\right].$$

In the concrete setting [6], we say that $\mathcal{MA}$ is strongly unforgeable under chosen-message attacks (or UF *secure*) if the UF-advantage of all forgers using reasonable resources and respecting the IV properties of the MAC is small.

Pseudorandomness. Let $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ be a MAC. For the definition of a MAC used in this chapter, we can still apply the notion of pseudorandomness [6, 36] from Section 2.2 if $\mathsf{IVSp}_{\mathcal{MA}} = \{\varepsilon\}$ and if the tagging algorithm is stateless and deterministic. Specifically, let D be an adversary with access to an oracle, and let $\mathcal{F}_{\mathcal{MA}}$ denote the

set of all functions from $\mathsf{MsgSp}_{\mathcal{MA}}$ to $\mathsf{TagSp}_{\mathcal{MA}}$. We define the PRF-*advantage* of PRF-*adversary* D as

$$\mathbf{Adv}^{\text{prf}}_{\mathcal{MA}}(D) = \Pr\left[\, K \xleftarrow{\$} \mathcal{K} \;:\; D^{\mathcal{T}^{\varepsilon}_{K}(\cdot)} = 1 \,\right] - \Pr\left[\, g \xleftarrow{\$} \mathcal{F}_{\mathcal{MA}} \;:\; D^{g(\cdot)} = 1 \,\right] .$$

In the concrete setting [6], we say that $\mathcal{MA}$ is a *secure pseudorandom function* (*PRF*) if the PRF-advantage of all PRF-adversaries using reasonable resources is small. If a MAC is PRF-secure, then it is also UF-secure [6].

Privacy. One can apply the PRIV-CPA notion of privacy for symmetric encryption schemes to MACs. Carter-Wegman MACs [81] and UMAC [22] are examples of MACs that preserve privacy.

We can extend the PRIV-DCPA definition of privacy under *distinct* chosen-plaintext attacks from Chapter 3 to MACs that expose their IVs to the caller. Specifically, let $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ be a message authentication scheme as defined in this chapter. Let $b \in \{0,1\}$ be a bit. Let A be an adversary with access to a left-or-right tagging oracle $\mathcal{T}_K(\cdot, \mathcal{LR}(\cdot,\cdot,b))$ that, on input $I \in \mathsf{IVSp}_{\mathcal{MA}}$, $M_0, M_1 \in \mathsf{MsgSp}_{\mathcal{MA}}$, returns $\mathcal{T}^I_K(M_b)$. We require that for all queries I, M_0, M_1 to the tagging oracle, $|M_0| = |M_1|$. If I^i, M^i_0, M^i_1 is the i-th oracle query, we require that for all indices j, k, $j \neq k$, $(I^j, M^j_0) \neq (I^k, M^k_0)$ and $(I^j, M^j_1) \neq (I^k, M^k_1)$; i.e., all left queries are distinct and all right queries are distinct. We call the adversary A *nonce-respecting* if it never queries its oracle with the same nonce twice. We define the *distinct-chosen-plaintext* (PRIV-DCPA) advantage of PRIV-DCPA-adversary A as

$$\begin{aligned} \mathbf{Adv}^{\text{priv-dcpa}}_{\mathcal{MA}}(A) \;\;=\;\; & \Pr\left[\, K \xleftarrow{\$} \mathcal{K} \;:\; A^{\mathcal{T}_K(\cdot,\mathcal{LR}(\cdot,\cdot,1))} = 1 \,\right] \\ & - \Pr\left[\, K \xleftarrow{\$} \mathcal{K} \;:\; A^{\mathcal{T}_K(\cdot,\mathcal{LR}(\cdot,\cdot,0))} = 1 \,\right] . \end{aligned}$$

We can also restate Theorem 3.7.4 from Chapter 3 for the type of (IV-exposing) message authentication schemes that we consider in this chapter. This theorem states that if a MAC $\mathcal{MA}$ has $\mathsf{IVSp}_{\mathcal{MA}} = \{\varepsilon\}$ is a secure PRF, then it is also PRIV-DCPA-secure.

Theorem 4.2.1 Let $\mathcal{MA}$ be a message authentication scheme with $\mathsf{IVSp}_{\mathcal{MA}} = \{\varepsilon\}$ and with a stateless and deterministic tagging algorithm. Then, given any PRIV-DCPA adversary A against $\mathcal{MA}$, we can construct a distinguisher D against $\mathcal{MA}$ such that

$$\mathbf{Adv}^{\text{priv-dcpa}}_{\mathcal{MA}}(A) \leq 2 \cdot \mathbf{Adv}^{\text{prf}}_{\mathcal{MA}}(D)$$

Furthermore, D uses the same resources of A. ∎

As in Chapter 3, even though many popular MACs are PRF-secure and the PRF notion implies the PRIV-DCPA notion, we consider the PRIV-DCPA notion because we wish to minimize the properties necessary in order to achieve our security goals.

4.3 Encoding Schemes

4.3.1 Overview

Our definition of encodings generalizes the encoding schemes in Chapter 3. An encoding scheme $\mathcal{EC}$ is an *un-keyed* public transformation that consists of four algorithms: Encode, DecodeA, DecodeB, and DecodeC. All algorithms may be stateful and Encode may be randomized. The decoding algorithms DecodeA, DecodeB, and DecodeC may share state. The specific properties of the algorithms, including their syntax, depend on the paradigm and the type of AEAD scheme in question. We first discuss some commonalities between the encoding schemes for the three composition paradigms and five AEAD types. Determining the appropriate consistency and security requirements for encoding schemes is one of the main aspects of this research; we discuss these in detail in Section 4.3.2. We take care to ensure that if an encoding scheme satisfies its consistency requirements, then a composite scheme built from this encoding scheme, an encryption scheme, and a MAC is an AEAD scheme that satisfies its corresponding consistency requirements.

Encoding for encryption. Algorithm Encode pre-processes the input messages M_a, M_s of an AEAD scheme's encryption algorithm. Specifically, on input M_a, M_s, Encode

outputs a 5-tuple $(M_p, M_o, M_n, M_e, M_t)$. Intuitively, M_p is cleartext data communicated with the ciphertext, M_o is the IV/nonce for use with the underlying encryption scheme, M_e is the input for the encryption scheme, M_n is the IV/nonce for use with the underlying MAC, and M_t is the input for the MAC. The different paradigms then use these five strings in slightly different ways and slightly different orders, as shown in Figures 4.1–4.3 and described in detail in Section 4.4.

Decoding and decrypting. The algorithms DecodeA, DecodeB, and DecodeC are used in an AEAD scheme's decryption process, which typically involves first invoking DecodeA on M_p to get back (at least) M_o. After the underlying encryption scheme's decryption algorithm uses M_o to recover the message M_e, the AEAD scheme invokes $\mathsf{DecodeB}(M_p, M_e)$ to recover (at least) M_a and M_s. If no errors occurred, then the AEAD scheme's decryption algorithm returns (M_a, M_s).

Errors may occur during the decryption process, however. For example, the algorithms DecodeA or DecodeB may return $\perp$, indicating that there was a decoding failure, perhaps upon detecting a replayed message. When DecodeA or DecodeB return $\perp$, the decryption algorithm does not accept the ciphertext. It may also be the case that DecodeA and DecodeB do not return $\perp$, but the MAC verification fails. When this occurs, the decryption algorithm invokes $\mathsf{DecodeC}(\perp)$. If the tag verification succeeds, the decryption algorithm invokes $\mathsf{DecodeC}(\top)$. By calling DecodeC in this way, the decryption algorithm tells the decoding algorithms whether the packet was accepted. These can then update their state, perhaps by incrementing a counter.

Respecting the IV properties of $\mathcal{SE}$ and $\mathcal{MA}$. The encryption scheme $\mathcal{SE}$ and the MAC $\mathcal{MA}$ that the Encode algorithm is combined with in a generalized Encode-then-$\{$E&M,MtE,EtM$\}$ construction may have certain IV requirements in order for them to be secure. Let $(M_a^1, M_s^1), (M_a^2, M_s^2), \ldots$ be a sequence of messages, let Encode begin in its initial state, and for $i = 1, 2, \ldots$ let $(M_p^i, M_o^i, M_n^i, M_e^i, M_t^i) = \mathsf{Encode}(M_a^i, M_s^i)$. We call an encoding scheme *nonce-respecting for encryption* if $M_o^i \neq M_o^j$ for all distinct i, j. We call an encoding scheme *nonce-respecting for MACing* if $M_n^i \neq M_n^j$ for all dis-

tinct i, j. An encoding scheme for use with generalized Encode-then-{E&M,EtM} constructions (resp., generalized Encode-then-MtE constructions) is *random-IV-respecting for encryption* if the encoding algorithm always picks the value M_o uniformly at random from $\mathsf{IVSp}_{\mathcal{SE}}$, and only does so after determining M_e (resp., M_e, M_n, and M_t). An encoding scheme is *length-based IV-respecting for encryption* with respect to some length-based IV-deriving function f if the first M_o value the encoding scheme generates is chosen according to the rules described above for random-IV-respecting encoding schemes, and all subsequent M_o values are generated according to f, the initial M_o value, and the lengths of the previous M_e values.

If the IV spaces are finite, then it is impossible to run a nonce-respecting encoding scheme on an infinite number of inputs. Therefore, we associate to any encoding scheme $\mathcal{EC}$ a parameter $\mathsf{MaxNum}_{\mathcal{EC}}$, and we assume that the encoding scheme is not invoked more than $\mathsf{MaxNum}_{\mathcal{EC}}$ times. In the above discussion and in the following sections, whenever we write "for $i = 1, 2, \ldots$, run Encode," we assume that the iterations stop before i gets larger than $\mathsf{MaxNum}_{\mathcal{EC}}$. We use the same convention when discussing AEAD schemes built from $\mathcal{EC}$.

4.3.2 Syntax, Consistency, and Security

In this section we describe the properties of Type 1–5 E&M, MtE and EtM encoding schemes. A consequence of the generality of our analyses is that the definitions of the consistency and security have a number of detailed sub-cases. Nevertheless, one can create natural encoding schemes that have these target consistency and security properties.

E&M encoding scheme syntax and consistency. Let $\mathcal{EC}^{\text{E\&M}} = (\mathsf{Encode}, \mathsf{DecodeA}, \mathsf{DecodeB}, \mathsf{DecodeC})$ be an E&M encoding scheme. The syntax of Encode is as described in Section 4.3.1. DecodeA, on input a string M_p, outputs a string M_o or $\perp$. DecodeB, on input two messages M_p, M_e, returns a 4-tuple of messages (M_a, M_s, M_n, M_t) or $(\perp, \perp, \perp, \perp)$. DecodeC takes as input the symbol $\top$ or the symbol $\perp$ and returns nothing.

Consider any two pairs of messages (M_a, M_s), (M_a, M'_s) with $|M_s| = |M'_s|$. Let $(M_p, M_o, M_n, M_e, M_t) \xleftarrow{\$} \mathsf{Encode}(M_a, M_s)$ for Encode in some state, and $(M'_p, M'_o, M'_n, M'_e, M'_t) \xleftarrow{\$} \mathsf{Encode}(M_a, M'_s)$ for Encode in some (possibly different) state. We require that $|M_e| = |M'_e|$ and $|M_t| = |M'_t|$. If this were not the case, the composite generalized Encode-then-E&M construction might not preserve privacy.

Consider also any two sequences of message pairs $(M^1_a, M^1_s), (M^2_a, M^2_s), \ldots$ and $(N^1_a, N^1_s), (N^2_a, N^2_s), \ldots$. Let Encode begin in its initial state and for $i = 1, 2, \ldots$ let $(M^i_p, M^i_o, M^i_n, M^i_e, M^i_t) = \mathsf{Encode}(M^i_a, M^i_s)$. Similarly, let Encode begin in its initial state and for $i = 1, 2, \ldots$ let $(N^i_p, N^i_o, N^i_n, N^i_e, N^i_t) = \mathsf{Encode}(N^i_a, N^i_s)$. If Encode is randomized, assume that both sequences are generated using the same random tape. Further assume that the randomness used in each invocation is recoverable from the output and that the amount of randomness used per invocation is a deterministic function of the lengths of the inputs. Consider any index i. If $|M^j_s| = |N^j_s|$ and $M^j_a = N^j_a$ for all $j \leq i$, then we require that $M^i_p = N^i_p$, $M^i_o = N^i_o$, and $M^i_n = N^i_n$.

Let $(M^1_a, M^1_s), (M^2_a, M^2_s), \ldots$ be a sequence of message pairs, and beginning with the encoder Encode in its initial state, let $(M^i_p, M^i_o, M^i_n, M^i_e, M^i_t) = \mathsf{Encode}(M^i_a, M^i_s)$ for $i = 1, 2, \ldots$. We make the following additional consistency requirements on $\mathcal{EC}^{\text{E\&M}}$, depending on the type of AEAD scheme in question. We use the notation $\mathsf{Decode}[\mathsf{ABC}]$ to denote any one of the decoding algorithms.

Type 1 For any i and for any state of the decoder, we require that $\mathsf{DecodeA}(M^i_p) = M^i_o$ and $\mathsf{DecodeB}(M^i_p, M^i_e) = (M^i_a, M^i_s, M^i_n, M^i_t)$.

Type 2 For any distinct indices i, j, we require that $(M^i_p, M^i_e) \neq (M^j_p, M^j_e)$. For any i, we require that for any state of the decoder, $\mathsf{DecodeA}(M^i_p) = M^i_o$. Furthermore, if DecodeB has not been invoked with (M^i_p, M^i_e) or if DecodeB has been invoked with (M^i_p, M^i_e) but for each such invocation the next call to $\mathsf{Decode}[\mathsf{ABC}]$ was $\mathsf{DecodeC}(\perp)$, then it must be the case that $\mathsf{DecodeB}(M^i_p, M^i_e) = (M^i_a, M^i_s, M^i_n, M^i_t)$.

Type 3 For any distinct indices i, j, we require that $(M^i_p, M^i_e) \neq (M^j_p, M^j_e)$. For any i,

we require that for any state of the decoder, $\mathsf{DecodeA}(M_p^i) = M_o^i$. Furthermore, if DecodeB has not been invoked with (M_p^j, M_e^j) for any $j \geq i$, or if DecodeB has been invoked with (M_p^j, M_e^j), for some $j \geq i$, but for each such invocation the next call to Decode[ABC] was $\mathsf{DecodeC}(\perp)$, then $\mathsf{DecodeB}(M_p^i, M_e^i) = (M_a^i, M_s^i, M_n^i, M_t^i)$.

Type 4 For $i = 1, 2, \ldots$ and with the decoder beginning in its initial state, let $m_o^i = \mathsf{DecodeA}(M_p^i)$ and $(m_a^i, m_s^i, m_n^i, m_t^i) = \mathsf{DecodeB}(M_p^i, M_e^i)$. We require that $M_a^i = m_a^i$, $M_s^i = m_s^i$, $M_o^i = m_o^i$, $M_n^i = m_n^i$, and $M_t^i = m_t^i$ for all i.

Type 5 We use the term E&M *calling sequence* to denote some sequence of calls to Decode[ABC] as they might appear in a generalized Encode-then-E&M construction. Namely, an E&M calling sequence consists of a call $\mathsf{DecodeA}(M_p)$ for some M_p and, if the response is not $\perp$, a call $\mathsf{DecodeB}(M_p, M_e)$ for some M_e, and, if the response is not $(\perp, \perp, \perp, \perp)$, a call to DecodeC. We say that (M_p, M_e) is *successfully decoded* if, in an E&M calling sequence, the responses of the first two decoding algorithms are not $\perp$ or $(\perp, \perp, \perp, \perp)$, respectively, and $\mathsf{DecodeC}(\top)$ is called.

Assume that the decoding algorithms are always called as per the E&M calling sequence (e.g., a DecodeB call is always followed by a DecodeC call unless DecodeB returns $(\perp, \perp, \perp, \perp)$). Fix $i \geq 0$ and assume that the messages that have been successfully decoded are $(M_p^1, M_e^1), \ldots, (M_p^i, M_e^i)$, and that they were decoded in order. We require that after invoking $\mathsf{DecodeA}(M_p^{i+1})$ followed by $\mathsf{DecodeB}(M_p^{i+1}, M_e^{i+1})$ and then $\mathsf{DecodeC}(\top)$, the response to the first call is M_o^{i+1} and the response to the second one is $(M_a^{i+1}, M_s^{i+1}, M_n^{i+1}, M_t^{i+1})$.

MtE encoding syntax and consistency. Let $\mathcal{EC}^{\text{MtE}} = (\mathsf{Encode}, \mathsf{DecodeA}, \mathsf{DecodeB}, \mathsf{DecodeC})$ be an MtE encoding scheme. The algorithms that constitute an MtE encoding scheme have the same syntax as those in an E&M encoding scheme.

Consider any two pairs of messages (M_a, M_s), (M_a, M_s'), where $|M_s| = |M_s'|$.

Let $(M_p, M_o, M_n, M_e, M_t) \xleftarrow{\$} \mathsf{Encode}(M_a, M_s)$ for Encode in some state, and $(M'_p, M'_o, M'_n, M'_e, M'_t) \xleftarrow{\$} \mathsf{Encode}(M_a, M'_s)$ for Encode is in some (possibly different) state. We require that $|M_e| = |M'_e|$. Consider also any two sequences of message pairs (M_a^1, M_s^1), $(M_a^2, M_s^2), \ldots$ and $(N_a^1, N_s^1), (N_a^2, N_s^2), \ldots$. Let Encode begin in its initial state and for $i = 1, 2, \ldots$ let $(M_p^i, M_o^i, M_n^i, M_e^i, M_t^i) = \mathsf{Encode}(M_a^i, M_s^i)$. Similarly, let Encode begin in its initial state and for $i = 1, 2, \ldots$ let $(N_p^i, N_o^i, N_n^i, N_e^i, N_t^i) = \mathsf{Encode}(N_a^i, N_s^i)$. If Encode is randomized, assume that both sequences are generated using the same random tape. Unlike with E&M encoding schemes, we do not require that the randomness used in each invocation be recoverable from the output. Consider any index i. If $|M_s^j| = |N_s^j|$ and $M_a^j = N_a^j$ for all $j \leq i$, then we require that $M_p^i = N_p^i$ and $M_o^i = N_o^i$.

The remainder of the consistency requirements for Type 1–Type 4 MtE encoding schemes are the same as those for the corresponding E&M encoding schemes. We make the following consistency requirement on encoding schemes for Type 5 CTs. We use the term MtE *calling sequence* to refer to some sequence of calls to Decode[ABC] as they might appear in a generalized Encode-then-MtE construction. Namely, an MtE calling sequence consists of a call $\mathsf{DecodeA}(M_p)$ and, if the response is not $\perp$, either a call $\mathsf{DecodeC}(\perp)$ finalizing the calling sequence, or a call $\mathsf{DecodeB}(M_p, M_e)$ for some M_e and, if the response is not $(\perp, \perp, \perp, \perp)$, a call to DecodeC. We say that (M_p, M_e) is *successfully decoded* if, in an MtE calling sequence, the responses of decoding algorithms DecodeA and DecodeB are not $\perp$ or $(\perp, \perp, \perp, \perp)$, respectively, and $\mathsf{DecodeC}(\perp)$ is never called.

Assume that the decoding algorithms are always called in successive MtE calling sequences. Fix $i \geq 0$ and assume that the messages that have been successfully decoded are $(M_p^1, M_e^1), \ldots, (M_p^i, M_e^i)$, and that they were decoded in order. We require that after invoking $\mathsf{DecodeA}(M_p^{i+1})$ followed by $\mathsf{DecodeB}(M_p^{i+1}, M_e^{i+1})$ and then $\mathsf{DecodeC}(\top)$, the response to the first call is M_o^{i+1} and the response to the second one is $(M_a^{i+1}, M_s^{i+1}, M_n^{i+1}, M_t^{i+1})$.

EtM encoding syntax and consistency. Let $\mathcal{EC}^{\text{EtM}} = (\mathsf{Encode}, \mathsf{DecodeA}, \mathsf{DecodeB}, \mathsf{DecodeC})$ be an EtM encoding scheme. The syntax of Encode is as described in Section 4.3.1. DecodeA, on input a string M_p, outputs a 3-tuple of messages (M_o, M_n, M_t) or $(\perp, \perp, \perp)$. DecodeB, on input two messages M_p, M_e, returns a pair of messages (M_a, M_s) or $(\perp, \perp)$. DecodeC takes as input the symbol $\top$ or the symbol $\perp$ and returns nothing.

Consider any two pairs of messages (M_a, M_s), (M_a, M'_s) with $|M_s| = |M'_s|$. Let $(M_p, M_o, M_n, M_e, M_t) \xleftarrow{\$} \mathsf{Encode}(M_a, M_s)$ for Encode in some state, and $(M'_p, M'_o, M'_n, M'_e, M'_t) \xleftarrow{\$} \mathsf{Encode}(M_a, M'_s)$ for Encode in some (possibly different) state. We require that $|M_e| = |M'_e|$. Consider also any two sequences of message pairs (M^1_a, M^1_s), $(M^2_a, M^2_s), \ldots$ and $(N^1_a, N^1_s), (N^2_a, N^2_s), \ldots$. For $i = 1, 2, \ldots$ let $(M^i_p, M^i_o, M^i_n, M^i_e, M^i_t) = \mathsf{Encode}(M^i_a, M^i_s)$ and $(N^i_p, N^i_o, N^i_n, N^i_e, N^i_t) = \mathsf{Encode}(N^i_a, N^i_s)$. Assume that each sequence is generated with Encode starting in its initial state. If Encode is randomized, assume that both sequences are generated using the same random tape. Consider any index i. If $|M^j_s| = |N^j_s|$ and $M^j_a = N^j_a$ for all $j \leq i$, then we require that $M^i_p = N^i_p$, $M^i_o = N^i_o$, $M^i_n = N^i_n$, and $M^i_t = N^i_t$.

We make the following additional consistency requirements on $\mathcal{EC}^{\text{EtM}}$, depending on the type of AEAD scheme in question. Let $(M^1_a, M^1_s), (M^2_a, M^2_s), \ldots$ be a sequence of messages and, beginning with Encode in its initial state, let $(M^i_p, M^i_o, M^i_n, M^i_e, M^i_t) = \mathsf{Encode}(M^i_a, M^i_s)$ for $i = 1, 2, \ldots$. We use the notation Decode[ABC] to denote any one of the decoding algorithms.

Type 1 For any i and for any state of the decoder, we require that $\mathsf{DecodeA}(M^i_p) = (M^i_o, M^i_n, M^i_t)$ and $\mathsf{DecodeB}(M^i_p, M^i_e) = (M^i_a, M^i_s)$.

Type 2 For any distinct indices i, j, we require that $(M^i_p, M^i_e) \neq (M^j_p, M^j_e)$. For any i, we require that for any state of the decoder, $\mathsf{DecodeA}(M^i_p) = (M^i_o, M^i_n, M^i_t)$. If DecodeB has not been invoked with (M^i_p, M^i_e) or if DecodeB has been invoked with (M^i_p, M^i_e) but for each such invocation the next call to Decode[ABC] was $\mathsf{DecodeC}(\perp)$, then $\mathsf{DecodeB}(M^i_p, M^i_e) = (M^i_a, M^i_s)$.

Type 3 For any distinct indices i, j, we require that $(M_p^i, M_e^i) \neq (M_p^j, M_e^j)$. For any i, we require that for any state of the decoder, $\mathsf{DecodeA}(M_p^i) = (M_o^i, M_n^i, M_t^i)$. Furthermore, if DecodeB has not been invoked with (M_p^j, M_e^j) for any $j \geq i$, or if DecodeB has been invoked with (M_p^j, M_e^j), for some $j \geq i$, but for each such invocation the next call to Decode[ABC] was $\mathsf{DecodeC}(\perp)$, then $\mathsf{DecodeB}(M_p^i, M_e^i) = (M_a^i, M_s^i)$.

Type 4 For $i = 1, 2, \ldots$ and the decoder beginning in its initial state, let $(m_o^i, m_n^i, m_t^i) = \mathsf{DecodeA}(M_p^i)$ and $(m_a^i, m_s^i) = \mathsf{DecodeB}(M_p^i, M_e^i)$. We require that $M_a^i = m_a^i$, $M_s^i = m_s^i$, $M_o^i = m_o^i$, $M_n^i = m_n^i$, and $M_t^i = m_t^i$ for all i.

Type 5 We use the term EtM *calling sequence* to refer to some sequence of calls to Decode[ABC] as they might appear in a generalized Encode-then-EtM construction. Note that they have the same form as MtE calling sequences. We say that (M_p, M_e) is *successfully decoded* if, in an EtM calling sequence, the responses of decoding algorithms DecodeA and DecodeB are not $(\perp, \perp, \perp)$ or $(\perp, \perp)$, respectively, and $\mathsf{DecodeC}(\perp)$ is never called.

Assume that the decoding algorithms are always called in successive EtM calling sequences. Fix $i \geq 0$ and assume that the messages that have been successfully decoded are $(M_p^1, M_e^1), \ldots, (M_p^i, M_e^i)$, and that they were decoded in order. We require that after invoking $\mathsf{DecodeA}(M_p^{i+1})$ followed by $\mathsf{DecodeB}(M_p^{i+1}, M_e^{i+1})$ and then $\mathsf{DecodeC}(\top)$, the response to the first call is $(M_o^{i+1}, M_n^{i+1}, M_t^{i+1})$ and the response to the second one is (M_a^{i+1}, M_s^{i+1}).

Security definitions. We now state our security definitions for E&M, MtE and EtM encoding schemes. We state the E&M and MtE definitions together because, for a given $n \in \{1, \ldots, 5\}$, the chosen-ciphertext security definition for a Type n E&M encoding scheme, E&M-SECn, is equivalent to the chosen-ciphertext security definition for a Type n MtE encoding scheme. The E&M-SEC4 definition is based to the COLL-CCA definition from Chapter 3. For E&M encoding schemes, we also state a chosen-plaintext

collision resistance definition, E&M-COLL, which is based on the definition COLL-CPA from Chapter 3.

Definition 4.3.1 (Security of E&M and MtE encoding schemes.) Let $\mathcal{EC} = (\mathsf{Encode}, \mathsf{DecodeA}, \mathsf{DecodeB}, \mathsf{DecodeC})$ be an E&M or an MtE encoding scheme. Let A_{cpa} be an adversary with access to an encoding oracle $\mathsf{Encode}(\cdot,\cdot)$ and for $n = 1, \ldots, 5$, let A_n be an adversary with access to an encoding oracle and decoding oracles $\mathsf{DecodeA}(\cdot)$, $\mathsf{DecodeB}(\cdot,\cdot)$, $\mathsf{DecodeC}(\cdot)$. Let (M_a^i, M_s^i) denote an adversary's i-th encoding query and let $(M_p^i, M_o^i, M_n^i, M_e^i, M_t^i)$ denote the response for that query. Let (m_p^i, m_e^i) denote A_n's i-th $\mathsf{DecodeB}(\cdot,\cdot)$ query and let $(m_a^i, m_s^i, m_n^i, m_t^i)$ denote the response for that query. Consider the following experiments.

Experiment $\mathbf{Exp}_{\mathcal{EC}}^{\text{e\&m-coll}}(A_{\text{cpa}})$

Run $A_{\text{cpa}}^{\mathsf{Encode}(\cdot,\cdot)}$ and if it makes two queries (M_a^i, M_s^i) and (M_a^j, M_s^j) to $\mathsf{Encode}(\cdot,\cdot)$ such that $i \neq j$ and $(M_n^i, M_t^i) = (M_n^j, M_t^j)$

then return 1 else return 0

Experiment $\mathbf{Exp}_{\mathcal{EC}}^{\text{e\&m-sec1}}(A_1)$

Run $A_1^{\mathsf{Encode}(\cdot,\cdot),\mathsf{DecodeA}(\cdot),\mathsf{DecodeB}(\cdot,\cdot),\mathsf{DecodeC}(\cdot)}$ and, if the following occurs:

— A_1 makes a query (M_a^i, M_s^i) to $\mathsf{Encode}(\cdot,\cdot)$ and a query (m_p^j, m_e^j) to $\mathsf{DecodeB}(\cdot,\cdot)$ such that $(M_p^i, M_e^i) \neq (m_p^j, m_e^j)$ and $(M_n^i, M_t^i) = (m_n^j, m_t^j)$

then return 1 else return 0

Experiment $\mathbf{Exp}_{\mathcal{EC}}^{\text{e\&m-sec2}}(A_2)$

Run $A_2^{\mathsf{Encode}(\cdot,\cdot),\mathsf{DecodeA}(\cdot),\mathsf{DecodeB}(\cdot,\cdot),\mathsf{DecodeC}(\cdot)}$ and, if one of the following occurs:

— A_2 makes a query (M_a^i, M_s^i) to $\mathsf{Encode}(\cdot,\cdot)$ and a query (m_p^j, m_e^j) to $\mathsf{DecodeB}(\cdot,\cdot)$ such that $(M_p^i, M_e^i) \neq (m_p^j, m_e^j)$ and $(M_n^i, M_t^i) = (m_n^j, m_t^j)$

— A_2 makes queries (m_p^j, m_e^j) and (m_p^k, m_e^k), where $(m_p^j, m_e^j) = (m_p^k, m_e^k)$ and $j \neq k$, to $\mathsf{DecodeB}(\cdot,\cdot)$ such that the next $\mathsf{Decode}[\mathsf{ABC}]$ query following the first of these queries is a call $\mathsf{DecodeC}(\top)$, and the response for the second of these queries is not $(\perp, \perp, \perp, \perp)$

then return 1 else return 0

Experiment $\mathbf{Exp}^{\text{e\&m-sec3}}_{\mathcal{EC}}(A_3)$

Run $A_3^{\mathsf{Encode}(\cdot,\cdot),\mathsf{DecodeA}(\cdot),\mathsf{DecodeB}(\cdot,\cdot),\mathsf{DecodeC}(\cdot)}$ and, if one of the following occurs:

— A_3 makes a query (M_a^i, M_s^i) to $\mathsf{Encode}(\cdot,\cdot)$ and a query (m_p^j, m_e^j) to $\mathsf{DecodeB}(\cdot,\cdot)$ such that $(M_p^i, M_e^i) \neq (m_p^j, m_e^j)$ and $(M_n^i, M_t^i) = (m_n^j, m_t^j)$

— A_3 makes queries (m_p^j, m_e^j) and (m_p^{j+l}, m_e^{j+l}), where $l \geq 1$, to $\mathsf{DecodeB}(\cdot,\cdot)$ such that the next $\mathsf{Decode[ABC]}$ query following the first of these queries is a call $\mathsf{DecodeC}(\top)$, the response for the second of these queries is not $(\perp,\perp,\perp,\perp)$, and for some i, k with $k \leq i$, $(m_p^j, m_e^j) = (M_p^i, M_e^i)$ and $(m_p^{j+l}, m_e^{j+l}) = (M_p^k, M_e^k)$

then return 1 else return 0

Experiment $\mathbf{Exp}^{\text{e\&m-sec4}}_{\mathcal{EC}}(A_4)$

Run $A_4^{\mathsf{Encode}(\cdot,\cdot),\mathsf{DecodeA}(\cdot),\mathsf{DecodeB}(\cdot,\cdot),\mathsf{DecodeC}(\cdot)}$ and, if one of the following occurs:

— A_4 makes a query (M_a^i, M_s^i) to $\mathsf{Encode}(\cdot,\cdot)$ and a query (m_p^j, m_e^j) to $\mathsf{DecodeB}(\cdot,\cdot)$ such that $i \neq j$ and $(M_n^i, M_t^i) = (m_n^j, m_t^j)$

— A_4 makes a query (M_a^j, M_s^j) to $\mathsf{Encode}(\cdot,\cdot)$ and a query (m_p^j, m_e^j) to $\mathsf{DecodeB}(\cdot,\cdot)$ such that $(M_p^j, M_e^j) \neq (m_p^j, m_e^j)$ and $(M_n^j, M_t^j) = (m_n^j, m_t^j)$

then return 1 else return 0

Experiment $\mathbf{Exp}^{\text{e\&m-sec5}}_{\mathcal{EC}}(A_5)$

Run $A_5^{\mathsf{Encode}(\cdot,\cdot),\mathsf{DecodeA}(\cdot),\mathsf{DecodeB}(\cdot,\cdot),\mathsf{DecodeC}(\cdot)}$ and, if one of the following occurs:

— A_5 makes a query (M_a^i, M_s^i) to $\mathsf{Encode}(\cdot,\cdot)$ and a query (m_p^j, m_e^j) to $\mathsf{DecodeB}(\cdot,\cdot)$ such that $(M_n^i, M_t^i) = (m_n^j, m_t^j)$ and, prior to the j-th $\mathsf{DecodeB}(\cdot,\cdot)$ query, A_5 did *not* make exactly $i-1$ $\mathsf{DecodeB}(\cdot,\cdot)$ queries that returned messages (i.e., not $\perp$) and that were followed by $\mathsf{DecodeC}(\top)$ calls

— A_5 makes a query (M_a^i, M_s^i) to $\mathsf{Encode}(\cdot,\cdot)$ and a query (m_p^j, m_e^j) to $\mathsf{DecodeB}(\cdot,\cdot)$ such that $(M_p^i, M_e^i) \neq (m_p^j, m_e^j)$ and $(M_n^i, M_t^i) = (m_n^j, m_t^j)$, and, prior to the j-th $\mathsf{DecodeB}(\cdot,\cdot)$ query, A_5 made *exactly* $i-1$ $\mathsf{DecodeB}(\cdot,\cdot)$ queries that returned messages (i.e., not $\perp$) and that were followed by $\mathsf{DecodeC}(\top)$ calls

then return 1 else return 0

The experiments $\mathbf{Exp}^{\text{mte-sec}n}_{\mathcal{EC}}(A_n)$ for MtE are identical to the $\mathbf{Exp}^{\text{e\&m-sec}n}_{\mathcal{EC}}(A_n)$ experiments for E&M.

We define the E&M-COLL-advantage of adversary A_{cpa}, and, for $n = 1, \ldots, 5$, the E&M-SECn-advantage and the MTE-SECn-advantage of adversary A_n, respectively, as follows:

$$\begin{aligned}
\mathbf{Adv}^{\text{e\&m-coll}}_{\mathcal{EC}}(A_{\text{cpa}}) &= \Pr\left[\,\mathbf{Exp}^{\text{e\&m-coll}}_{\mathcal{EC}}(A_{\text{cpa}}) = 1\,\right] \\
\mathbf{Adv}^{\text{e\&m-sec}n}_{\mathcal{EC}}(A_n) &= \Pr\left[\,\mathbf{Exp}^{\text{e\&m-sec}n}_{\mathcal{EC}}(A_n) = 1\,\right] \\
\mathbf{Adv}^{\text{mte-sec}n}_{\mathcal{EC}}(A_n) &= \Pr\left[\,\mathbf{Exp}^{\text{mte-sec}n}_{\mathcal{EC}}(A_n) = 1\,\right].
\end{aligned}$$

In the concrete setting [6], we say that an E&M encoding scheme $\mathcal{EC}$ is E&M-COLL-secure if $\mathbf{Adv}^{\text{e\&m-coll}}_{\mathcal{EC}}(A_{\text{cpa}})$ is small for all adversaries A_{cpa} using reasonable resources. For $n = 1, \ldots, 5$, we say that a Type n E&M encoding scheme $\mathcal{EC}$ is E&M-SECn-secure if $\mathbf{Adv}^{\text{e\&m-sec}n}_{\mathcal{EC}}(A_n)$ is small for all adversaries A_n using reasonable resources. For $n = 1, \ldots, 5$, we say that a Type n MtE encoding scheme $\mathcal{EC}$ is MTE-SECn-secure if $\mathbf{Adv}^{\text{mte-sec}n}_{\mathcal{EC}}(A_n)$ is small for all adversaries A_n using reasonable resources. ∎

Definition 4.3.2 (Security of EtM encoding schemes.) Consider an EtM encoding scheme $\mathcal{EC} = (\mathsf{Encode}, \mathsf{DecodeA}, \mathsf{DecodeB}, \mathsf{DecodeC})$. For $n = 1, \ldots, 5$, let A_n be an adversary with access to an encoding oracle $\mathsf{Encode}(\cdot,\cdot)$ and decoding oracles $\mathsf{DecodeA}(\cdot)$, $\mathsf{DecodeB}(\cdot,\cdot)$, $\mathsf{DecodeC}(\cdot)$. Let (M_a^i, M_s^i) denote an adversary's i-th encoding query and let $(M_p^i, M_o^i, M_n^i, M_e^i, M_t^i)$ denote the response for that query. Let m_p^i denote A_n's i-th $\mathsf{DecodeA}(\cdot)$ query and let (m_o^i, m_n^i, m_t^i) denote the response for that query. Consider the following experiments.

Experiment $\mathbf{Exp}^{\text{etm-sec1}}_{\mathcal{EC}}(A_1)$

Run $A_1^{\mathsf{Encode}(\cdot,\cdot),\mathsf{DecodeA}(\cdot),\mathsf{DecodeB}(\cdot,\cdot),\mathsf{DecodeC}(\cdot)}$ and, if the following occurs:

— A_1 makes a query (M_a^i, M_s^i) to $\mathsf{Encode}(\cdot,\cdot)$ and a query m_p^j to $\mathsf{DecodeA}(\cdot)$ such that $M_p^i \neq m_p^j$ and $(M_n^i, M_t^i) = (m_n^j, m_t^j)$

then return 1 else return 0

Experiment $\mathbf{Exp}^{\text{etm-sec2}}_{\mathcal{EC}}(A_2)$

Run $A_2^{\mathsf{Encode}(\cdot,\cdot),\mathsf{DecodeA}(\cdot),\mathsf{DecodeB}(\cdot,\cdot),\mathsf{DecodeC}(\cdot)}$ and, if one of the following occurs:

— A_2 makes a query (M_a^i, M_s^i) to $\mathsf{Encode}(\cdot,\cdot)$ and a query m_p^j to $\mathsf{DecodeA}(\cdot)$ such that $M_p^i \neq m_p^j$ and $(M_n^i, M_t^i) = (m_n^j, m_t^j)$

— A_2 makes queries (m_p^j, m_e^j) and (m_p^k, m_e^k), where $(m_p^j, m_e^j) = (m_p^k, m_e^k)$ and $j \neq k$, to $\mathsf{DecodeB}(\cdot,\cdot)$ such that the next $\mathsf{Decode}[\mathsf{ABC}]$ query following the first of these queries is a call $\mathsf{DecodeC}(\top)$, and the response for the second of these queries is not $(\perp, \perp)$

then return 1 else return 0

Experiment $\mathbf{Exp}^{\text{etm-sec3}}_{\mathcal{EC}}(A_3)$

Run $A_3^{\mathsf{Encode}(\cdot,\cdot),\mathsf{DecodeA}(\cdot),\mathsf{DecodeB}(\cdot,\cdot),\mathsf{DecodeC}(\cdot)}$ and, if one of the following occurs:

— A_3 makes a query (M_a^i, M_s^i) to $\mathsf{Encode}(\cdot,\cdot)$ and a query m_p^j to $\mathsf{DecodeA}(\cdot)$ such that $M_p^i \neq m_p^j$ and $(M_n^i, M_t^i) = (m_n^j, m_t^j)$

— A_3 makes queries (m_p^j, m_e^j) and (m_p^{j+l}, m_e^{j+l}), where $l \geq 1$, to $\mathsf{DecodeB}(\cdot,\cdot)$ such that the next $\mathsf{Decode}[\mathsf{ABC}]$ query following the first of these queries is a call $\mathsf{DecodeC}(\top)$, the response for the second of these queries is not $(\perp, \perp)$, and for some i, k with $k \leq i$, $(m_p^j, m_e^j) = (M_p^i, M_e^i)$ and $(m_p^{j+l}, m_e^{j+l}) = (M_p^k, M_e^k)$

then return 1 else return 0

Experiment $\mathbf{Exp}^{\text{etm-sec4}}_{\mathcal{EC}}(A_4)$

Run $A_4^{\mathsf{Encode}(\cdot,\cdot),\mathsf{DecodeA}(\cdot),\mathsf{DecodeB}(\cdot,\cdot),\mathsf{DecodeC}(\cdot)}$ and, if one of the following

occurs:

— A_4 makes a query (M_a^i, M_s^i) to $\mathsf{Encode}(\cdot,\cdot)$ and a query m_p^j to $\mathsf{DecodeA}(\cdot)$ such that $i \neq j$ and $(M_n^i, M_t^i) = (m_n^j, m_t^j)$

— A_4 makes a query (M_a^j, M_s^j) to $\mathsf{Encode}(\cdot,\cdot)$ and a query m_p^j to $\mathsf{DecodeA}(\cdot)$ such that $M_p^j \neq m_p^j$ and $(M_n^j, M_t^j) = (m_n^j, m_t^j)$

then return 1 else return 0

Experiment $\mathbf{Exp}^{\text{etm-sec5}}_{\mathcal{EC}}(A_5)$

Run $A_5^{\mathsf{Encode}(\cdot,\cdot),\mathsf{DecodeA}(\cdot),\mathsf{DecodeB}(\cdot,\cdot),\mathsf{DecodeC}(\cdot)}$ and, if A_5 only invokes $\mathsf{Decode}[\mathsf{ABC}]$ in legitimate EtM calling sequences, and one of the following occurs:

— A_5 makes a query (M_a^i, M_s^i) to $\mathsf{Encode}(\cdot,\cdot)$ and a query m_p^j to $\mathsf{DecodeA}(\cdot)$ such that $(M_n^i, M_t^i) = (m_n^j, m_t^j)$ and, prior to the j-th $\mathsf{DecodeA}(\cdot)$ query, A_5 did *not* make exactly $i-1$ $\mathsf{Decode}[\mathsf{ABC}]$ calling sequences that ended in the call $\mathsf{DecodeC}(\top)$

— A_5 makes a query (M_a^i, M_s^i) to $\mathsf{Encode}(\cdot,\cdot)$ and a query m_p^j to $\mathsf{DecodeA}(\cdot)$ such that $M_p^i \neq m_p^j$ and $(M_n^i, M_t^i) = (m_n^j, m_t^j)$, and, prior to the j-th $\mathsf{DecodeA}(\cdot)$ query, A_5 made *exactly* $i-1$ $\mathsf{Decode}[\mathsf{ABC}]$ calling sequences that ended in the call $\mathsf{DecodeC}(\top)$

then return 1 else return 0

For $n = 1, \ldots, 5$, we define the ETM-SECn-advantage of adversary A_n as

$$\mathbf{Adv}^{\text{etm-sec}n}_{\mathcal{EC}}(A_n) = \Pr\left[\, \mathbf{Exp}^{\text{etm-sec}n}_{\mathcal{EC}}(A_n) = 1 \,\right] .$$

In the concrete setting [6], for $n = 1, \ldots, 5$, we say that a Type n EtM encoding scheme $\mathcal{EC}$ is ETM-SECn-secure if $\mathbf{Adv}^{\text{etm-sec}n}_{\mathcal{EC}}(A_n)$ is small for all adversaries A_n using reasonable resources. ∎

4.4 Composition Methods

Having defined the syntax we use in this chapter for encryption and message authentication schemes and having presented our new definitions of encoding schemes, we are now in a position to formally state our generalized composition paradigms. Recall also Figures 4.1–4.3.

Construction 4.4.1 (Generalized Encode-then-E&M.) Let $\mathcal{EC}^{\text{E\&M}} = (\mathsf{Encode}, \mathsf{DecodeA}, \mathsf{DecodeB}, \mathsf{DecodeC})$, $\mathcal{SE} = (\mathcal{K}_e, \mathcal{E}, \mathcal{D})$, and $\mathcal{MA} = (\mathcal{K}_t, \mathcal{T}, \mathcal{V})$ be E&M encoding, encryption, and message authentication schemes, respectively, with compatible message spaces (e.g., the outputs from Encode are suitable inputs to $\mathcal{E}$ and $\mathcal{T}$). Let all states initially be ε. We associate to these schemes a *generalized Encode-then-E&M AEAD scheme* $\mathcal{AE}^{\text{E\&M}} = (\overline{\mathcal{K}}, \overline{\mathcal{E}}, \overline{\mathcal{D}})$ defined as follows:

Algorithm $\overline{\mathcal{K}}$

$K_e \xleftarrow{\$} \mathcal{K}_e$; $K_t \xleftarrow{\$} \mathcal{K}_t$

Return $\langle K_e, K_t \rangle$

Algorithm $\overline{\mathcal{E}}_{\langle K_e, K_t \rangle}(M_a, M_s)$

$(M_p, M_o, M_n, M_e, M_t) \xleftarrow{\$} \mathsf{Encode}(M_a, M_s)$

$\sigma \xleftarrow{\$} \mathcal{E}_{K_e}^{M_o}(M_e)$; $\tau \xleftarrow{\$} \mathcal{T}_{K_t}^{M_n}(M_t)$

Return $\langle M_p, \sigma, \tau \rangle$

Algorithm $\overline{\mathcal{D}}_{\langle K_e, K_t \rangle}(C)$

$\boxed{\text{If } st = \bot \text{ then return } (\bot, \bot)}$

If there does not exist M_p, σ, τ s.t. $C = \langle M_p, \sigma, \tau \rangle$ then

$\boxed{st \leftarrow \bot\ ;}$ return $(\bot, \bot)$

Parse C as $\langle M_p, \sigma, \tau \rangle$; $M_o \leftarrow \mathsf{DecodeA}(M_p)$

If $M_o = \bot$ then $\boxed{st \leftarrow \bot\ ;}$ return $(\bot, \bot)$

$M_e \leftarrow \mathcal{D}_{K_e}^{M_o}(\sigma)$

$(M_a, M_s, M_n, M_t) \leftarrow \mathsf{DecodeB}(M_p, M_e)$

If $M_s = \bot$ then $\boxed{st \leftarrow \bot\ ;}$ return $(\bot, \bot)$

$v \leftarrow \mathcal{V}_{K_t}^{M_n}(M_t, \tau)$

If $v = 0$ then $\boxed{st \leftarrow \perp\ ;}$ $\mathsf{DecodeC}(\perp)$; return $(\perp, \perp)$

$\mathsf{DecodeC}(\top)$

Return (M_a, M_s)

Type 4 AEAD schemes include the boxed portions of the above pseudocode and the other types do not. Recall that $\langle a_1, \ldots, a_m \rangle$ denotes an encoding of the strings $a_1, \ldots, a_m$ such that $a_1, \ldots, a_m$ are recoverable. For the call to $\mathsf{DecodeB}(M_p, M_e)$, recall that if any one of M_a, M_s, M_n, M_t is $\perp$, then they are all $\perp$. Although only $\overline{\mathcal{D}}$ explicitly maintains state in the above pseudocode, the underlying encoding, encryption, and MAC schemes may also maintain state. ▮

Construction 4.4.2 (Generalized Encode-then-MtE.) Let $\mathcal{EC}^{\text{MtE}} = (\mathsf{Encode}, \mathsf{DecodeA}, \mathsf{DecodeB}, \mathsf{DecodeC})$, $\mathcal{SE} = (\mathcal{K}_e, \mathcal{E}, \mathcal{D})$, and $\mathcal{MA} = (\mathcal{K}_t, \mathcal{T}, \mathcal{V})$, respectively, be MtE encoding, encryption, and message authentication schemes with compatible message spaces. Assume that $\mathcal{T}$ always produces tags of the same length. Let all states initially be ε. We associate to these schemes a *generalized Encode-then-MtE AEAD scheme* $\mathcal{AE}^{\text{MtE}} = (\overline{\mathcal{K}}, \overline{\mathcal{E}}, \overline{\mathcal{D}})$ defined as follows:

Algorithm $\overline{\mathcal{K}}$

$K_e \xleftarrow{\$} \mathcal{K}_e$; $K_t \xleftarrow{\$} \mathcal{K}_t$

Return $\langle K_e, K_t \rangle$

Algorithm $\overline{\mathcal{E}}_{\langle K_e, K_t \rangle}(M_a, M_s)$

$(M_p, M_o, M_n, M_e, M_t) \xleftarrow{\$} \mathsf{Encode}(M_a, M_s)$

$\tau \xleftarrow{\$} \mathcal{T}_{K_t}^{M_n}(M_t)$; $\sigma \xleftarrow{\$} \mathcal{E}_{K_e}^{M_o}(\langle M_e, \tau \rangle)$

Return $\langle M_p, \sigma \rangle$

Algorithm $\overline{\mathcal{D}}_{\langle K_e, K_t \rangle}(C)$

$\boxed{\text{If } st = \perp \text{ then return } (\perp, \perp)}$

If there does not exist M_p, σ s.t. $C = \langle M_p, \sigma \rangle$ then $\boxed{st \leftarrow \perp\ ;}$ return $(\perp, \perp)$

Parse C as $\langle M_p, \sigma \rangle$; $M_o \leftarrow \mathsf{DecodeA}(M_p)$

If $M_o = \perp$ then $\boxed{st \leftarrow \perp\ ;}$ return $(\perp, \perp)$

$M \leftarrow \mathcal{D}_{K_e}^{M_o}(\sigma)$

If there does not exist M_e, τ s.t. $M = \langle M_e, \tau \rangle$ then

$\boxed{st \leftarrow \perp\ ;}$ $\mathsf{DecodeC}(\perp)$; return $(\perp, \perp)$

Parse M as $\langle M_e, \tau \rangle$

$(M_a, M_s, M_n, M_t) \leftarrow \mathsf{DecodeB}(M_p, M_e)$

If $M_s = \perp$ then $\boxed{st \leftarrow \perp\ ;}$ return $(\perp, \perp)$

$v \leftarrow \mathcal{V}_{K_t}^{M_n}(M_t, \tau)$

If $v = 0$ then $\boxed{st \leftarrow \perp\ ;}$ $\mathsf{DecodeC}(\perp)$; return $(\perp, \perp)$

$\mathsf{DecodeC}(\top)$

Return (M_a, M_s)

Type 4 AEAD schemes include the boxed portions of the above pseudocode and the other types do not. We require that the length of the string $\langle M_e, \tau \rangle$ depend only on the lengths of M_e and τ. ▌

Construction 4.4.3 (Generalized Encode-then-EtM.) Let $\mathcal{EC}^{\text{EtM}} = (\mathsf{Encode}, \mathsf{DecodeA}, \mathsf{DecodeB}, \mathsf{DecodeC})$, $\mathcal{SE} = (\mathcal{K}_e, \mathcal{E}, \mathcal{D})$, and $\mathcal{MA} = (\mathcal{K}_t, \mathcal{T}, \mathcal{V})$, respectively, be EtM encoding, encryption, and message authentication schemes with compatible message spaces. Let all states initially be ε. We associate to these schemes a *generalized Encode-then-EtM AEAD scheme* $\mathcal{AE}^{\text{EtM}} = (\overline{\mathcal{K}}, \overline{\mathcal{E}}, \overline{\mathcal{D}})$ defined as follows:

Algorithm $\overline{\mathcal{K}}$

$K_e \xleftarrow{\$} \mathcal{K}_e$; $K_t \xleftarrow{\$} \mathcal{K}_t$

Return $\langle K_e, K_t \rangle$

Algorithm $\overline{\mathcal{E}}_{\langle K_e, K_t \rangle}(M_a, M_s)$

$(M_p, M_o, M_n, M_e, M_t) \xleftarrow{\$} \mathsf{Encode}(M_a, M_s)$

$\sigma \xleftarrow{\$} \mathcal{E}_{K_e}^{M_o}(M_e)$; $\tau \xleftarrow{\$} \mathcal{T}_{K_t}^{M_n}(\langle M_t, \sigma \rangle)$

$C \leftarrow \langle M_p, \sigma, \tau \rangle$

Return C

Algorithm $\overline{\mathcal{D}}_{\langle K_e, K_t\rangle}(C)$

$\boxed{\text{If } st = \bot \text{ then return } (\bot, \bot)}$

If there does not exist M_p, σ, τ s.t. $C = \langle M_p, \sigma, \tau\rangle$ then

$\boxed{st \leftarrow \bot\ ;}$ return $(\bot, \bot)$

Parse C as $\langle M_p, \sigma, \tau\rangle$; $(M_o, M_n, M_t) \leftarrow \mathsf{DecodeA}(M_p)$

If $M_o = \bot$ then $\boxed{st \leftarrow \bot\ ;}$ return $(\bot, \bot)$

$v \leftarrow \mathcal{V}_{K_t}^{M_n}(\langle M_t, \sigma\rangle, \tau)$

If $v = 0$ then $\boxed{st \leftarrow \bot\ ;}$ $\mathsf{DecodeC}(\bot)$; return $(\bot, \bot)$

$M_e \leftarrow \mathcal{D}_{K_e}^{M_o}(\sigma)$

$(M_a, M_s) \leftarrow \mathsf{DecodeB}(M_p, M_e)$

If $M_s = \bot$ then $\boxed{st \leftarrow \bot\ ;}$ return $(\bot, \bot)$

$\mathsf{DecodeC}(\top)$

Return (M_a, M_s)

Type 4 AEAD schemes include the boxed portions of the above pseudocode and the other types do not. ∎

4.5 Generalized Encode-then-E&M Security

4.5.1 Privacy

Theorem 4.5.1 below captures our chosen-plaintext privacy result for generalized Encode-then-E&M AEAD constructions. Informally, this theorem states that if a Type n, $n \in \{1, \ldots, 5\}$, generalized Encode-then-E&M construction $\mathcal{AE}$ is built from an encryption scheme $\mathcal{SE}$, a MAC $\mathcal{MA}$, and a Type n E&M encoding scheme $\mathcal{EC}$, and if the latter respects the nonce requirements of $\mathcal{SE}$ and $\mathcal{MA}$, then $\mathcal{AE}$ will be PRIV-CPA-secure if (1) $\mathcal{SE}$ is PRIV-CPA-secure, $\mathcal{MA}$ is PRIV-DCPA-secure, and $\mathcal{EC}$ is E&M-COLL-secure or (2) $\mathcal{SE}$ is PRIV-CPA-secure and $\mathcal{MA}$ is PRIV-CPA-secure. We remark that if the underlying MAC requires a nonce, then $\mathcal{EC}$ is automatically E&M-COLL-secure ($\mathbf{Adv}_{\mathcal{EC}}^{\text{e\&m-coll}}(C) = 0$). We also recall that some MACs, e.g., Carter-Wegman MACs [81]

like UMAC [22] are PRIV-CPA-secure, and that PRF-secure MACs like OMAC [41] are also PRIV-DCPA-secure via Theorem 4.2.1.

Theorem 4.5.1 (Privacy of Generalized Encode-then-E&M Schemes.) Let $\mathcal{SE}$, $\mathcal{MA}$, and $\mathcal{EC}$ be an encryption, a message authentication, and an E&M encoding scheme, respectively. Let $\mathcal{AE}$ be the AEAD scheme associated to them as per Construction 4.4.1. Then, given any adversary S against $\mathcal{AE}$, there exist adversaries A, B, D, and C such that

$$\begin{aligned}\mathbf{Adv}^{\text{priv-cpa}}_{\mathcal{AE}}(S) &\leq \mathbf{Adv}^{\text{priv-cpa}}_{\mathcal{SE}}(A) + \mathbf{Adv}^{\text{priv-dcpa}}_{\mathcal{MA}}(D) + 2 \cdot \mathbf{Adv}^{\text{e\&m-coll}}_{\mathcal{EC}}(C) \quad \text{and} \\ \mathbf{Adv}^{\text{priv-cpa}}_{\mathcal{AE}}(S) &\leq \mathbf{Adv}^{\text{priv-cpa}}_{\mathcal{SE}}(A) + \mathbf{Adv}^{\text{priv-cpa}}_{\mathcal{MA}}(B)\ .\end{aligned}$$

Furthermore, $A, B, D,$ and C use the same resources as S except that A's, B's, and D's inputs to their respective oracles may be of different lengths than those of S (due to the encoding). If $\mathcal{EC}$ is nonce-respecting-for-encryption (resp., length-based IV-respecting-for-encryption or random-IV-respecting-for-encryption), then A will be nonce-respecting (resp., length-based IV-respecting or random-IV-respecting). Similarly, if $\mathcal{EC}$ is nonce-respecting-for-MACing, then B and D will be nonce-respecting. ∎

The proof of Theorem 4.5.1 is similar to the proof of Theorem 3.7.5 from Chapter 3; we omit details. The principle differences between the proof of Theorem 4.5.1 and the proof of Theorem 3.7.5 are the following: we consider AEAD schemes that take associated data; we allow $\mathcal{SE}$ to take nonces, length-based IVs, or random-IVs as input, and $\mathcal{MA}$ to take nonces as input; in order to use the hybrid argument, we exploit the fact that we can by definition recover the randomness from the output of $\mathcal{EC}$'s encoding function; because the encoding algorithm controls the IVs for the underlying encryption scheme and MAC, we use the same randomness for both encoding sequences.

4.5.2 Integrity

We begin by formalizing a new property for generalized Encode-then-E&M AEAD schemes. As with our use of the PRIV-DCPA notion, we use this security notion because we believe it important to accurately describe the specific properties that

we require from the AEAD scheme. In most situations, however, one does not actually need to manipulate this definition but must merely invoke Proposition 4.5.3, which states that if an AEAD scheme's underlying encryption algorithm is length-preserving, then the AEAD scheme automatically has the property that we specify below.

Definition 4.5.2 Fix $n \in \{1, \ldots, 5\}$. Let $\mathcal{SE}$, $\mathcal{MA}$, and $\mathcal{EC}$, respectively, be an encryption, a message authentication, and an E&M encoding scheme. Let $\mathcal{AE} = (\overline{\mathcal{K}}, \overline{\mathcal{E}}, \overline{\mathcal{D}})$ be a Type n AEAD scheme associated to them as per Construction 4.4.1. Let A be an adversary with access to an encryption oracle $\overline{\mathcal{E}}_K(\cdot, \cdot)$ and a decryption oracle $\overline{\mathcal{D}}_K(\cdot)$. Let (M_a^i, M_s^i) denote the adversary's i-th encryption oracle query, $(M_p^i, M_o^i, M_n^i, M_e^i, M_t^i)$ denote the encoding of that query, and $\langle M_p^i, \sigma_i, \tau_i \rangle$ denote the returned ciphertext. Let $\langle m_p^i, \sigma_i', \tau_i' \rangle$ denote the i-th decryption query (assuming it is parsable), and $m_o^i, m_n^i, m_e^i, m_t^i, m_a^i, m_s^i$ denote the internal values in the decryption process (or $\perp$ if an error occurs during decryption). A "wins" if it makes a decryption query $\langle m_p^j, \sigma_j', \tau_j' \rangle$ such that $(m_o^j, m_e^j) = (M_o^i, M_e^i)$ for some $i \in \{1, \ldots, k\}$ but $\sigma_j' \neq \sigma_i$ (where k is the number of $\overline{\mathcal{E}}_K(\cdot, \cdot)$ oracle queries made by A before A's j-th decryption query). We define the E&M-SP-advantage of E&M-SP adversary A as

$$\mathbf{Adv}^{\text{e\&m-sp}}_{\mathcal{AE}}(A) = \Pr\left[K \xleftarrow{\$} \overline{\mathcal{K}} \ : \ A \text{ "wins"} \right] . \ \blacksquare$$

The following proposition shows that if the underlying encryption scheme is length preserving, then an adversary cannot win the game described in the above definition.

Proposition 4.5.3 Fix $n \in \{1, \ldots, 5\}$. Let $\mathcal{SE}$, $\mathcal{MA}$, and $\mathcal{EC}$, respectively, be an encryption, a MAC, and a Type n E&M encoding scheme. Let $\mathcal{AE} = (\overline{\mathcal{K}}, \overline{\mathcal{E}}, \overline{\mathcal{D}})$ be a Type n AEAD scheme associated to them as per Construction 4.4.1. Let A be an E&M-SP-adversary. If $\mathcal{SE}$'s encryption operation is length-preserving, then

$$\mathbf{Adv}^{\text{e\&m-sp}}_{\mathcal{AE}}(A) = 0 \ . \ \blacksquare$$

Proof: If $\mathcal{SE}$'s encryption operation is length-preserving, then given any IV I, the encryption operation is bijective. This means A can never win. $\blacksquare$

We now state our authenticity result for generalized Encode-then-E&M constructions. Informally, this theorem states that if a Type n, $n \in \{1, \ldots, 5\}$, generalized Encode-then-E&M construction $\mathcal{AE}$ is built from an encryption scheme $\mathcal{SE}$, a MAC $\mathcal{MA}$, and a Type n E&M encoding scheme $\mathcal{EC}$, and if the latter respects the nonce requirements of $\mathcal{SE}$ and $\mathcal{MA}$, then $\mathcal{AE}$ will be AUTHn-secure if $\mathcal{MA}$ is UF-secure, $\mathcal{EC}$ is E&M-SECn-secure, and $\mathcal{AE}$ has the E&M-SP property specified above. As Proposition 4.5.3 shows, it is easy to construct AEAD schemes that have the E&M-SP property. We further remark that while the definitions for E&M-SECn-security may be involved, with multiple subcases, there exist natural encoding schemes that satisfy the E&M-SECn security definitions.

Theorem 4.5.4 (Integrity of Generalized Encode-then-E&M Schemes.) Fix $n \in \{1, \ldots, 5\}$. Let $\mathcal{SE}$, $\mathcal{MA}$, and $\mathcal{EC}$, respectively, be an encryption, a MAC, and a Type n E&M encoding scheme. Let $\mathcal{AE}$ be a Type n AEAD scheme associated to them as per Construction 4.4.1. Then, given any AUTHn-adversary I against $\mathcal{AE}$, there exist adversaries F, C, and S such that

$$\mathbf{Adv}^{\text{auth}n}_{\mathcal{AE}}(I) \leq \mathbf{Adv}^{\text{uf}}_{\mathcal{MA}}(F) + \mathbf{Adv}^{\text{e\&m-sec}n}_{\mathcal{EC}}(C) + \mathbf{Adv}^{\text{e\&m-sp}}_{\mathcal{AE}}(S) \ .$$

Furthermore, F, C, and S use the same resources as I except that F's messages to its oracles may be of different lengths than I's queries to its oracles (due to encoding) and C's messages to its decoding oracle may have slightly different lengths than I's decryption queries. If $\mathcal{EC}$ is nonce-respecting-for-MACing, then F will be nonce-respecting. ∎

The proof of the above theorem is below. The proof for Type 4 AEAD schemes is similar to the proof of Theorem 3.8.2 in Chapter 3 except that here we consider AEAD schemes that handle associated data.

Proof of Theorem 4.5.4: Let F, C, and S be adversaries that run I and reply to I's oracle queries using their own oracles. In more detail, F presents I with encryption and decryption-verification oracles exactly as in Construction 4.4.1 except that F uses its own oracles for handling the tagging and verification portions of Construction 4.4.1. Similarly, C runs I exactly as in Construction 4.4.1 except that it runs all encoding and

decoding operations through its own oracles. In the case of S, S simply passes all of I's encryption and decryption queries to its (S's) own oracles.

Let (M_a^i, M_s^i) denote I's i-th oracle query, let $(M_p^i, M_o^i, M_n^i, M_e^i, M_t^i)$ denote the encoding of that query, and let $\langle M_p^i, \sigma_i, \tau_i \rangle$ denote the returned ciphertext. Additionally, let $\langle m_p^i, \sigma_i', \tau_i' \rangle$ denote the i-th decryption-verification query (assuming it is parsable), and $m_o^i, m_n^i, m_e^i, m_t^i, m_a^i, m_s^i$ denote the internal values in the decryption process (or $\perp$ if an error occurs during decryption). Let j denote the index of I's (first) winning query and let k denote the number of encryption oracle queries performed at the time I wins.

Let E be the event that I wins. By partitioning event E, we will see that if I succeeds in forging, then one of F, C, and S also wins their game.

For a Type 1 AEAD scheme, we partition event E as follows:

$$
\begin{aligned}
E \quad &: I \text{ wins} \\
E_1 \quad &: E \text{ occurs and } (m_p^j, m_e^j, \tau_j') \in \{\, (M_p^i, M_e^i, \tau_i) \;:\; 1 \le i \le k \,\} \;\; /\!/ \; S \text{ wins} \\
E_2 \quad &: E \text{ occurs and } (m_p^j, m_e^j, \tau_j') \notin \{\, (M_p^i, M_e^i, \tau_i) \;:\; 1 \le i \le k \,\} \\
E_{2,1} \quad &: E_2 \text{ occurs and } (m_n^j, m_t^j, \tau_j') \notin \{\, (M_n^i, M_t^i, \tau_i) \;:\; 1 \le i \le k \,\} \;\; /\!/ \; F \text{ wins} \\
E_{2,2} \quad &: E_2 \text{ occurs and } (m_n^j, m_t^j, \tau_j') \in \{\, (M_n^i, M_t^i, \tau_i) \;:\; 1 \le i \le k \,\} \;\; /\!/ \; C \text{ wins}
\end{aligned}
$$

The above partitioning shows that if event E occurs, then one of E_1, $E_{2,1}$, or $E_{2,2}$ must occur. Note that if E_1 occurs then S wins its game. This is because $m_p^j = M_p^i$ (and therefore $m_o^j = M_o^i$ by consistency requirements on the encoding scheme) and $\tau_j' = \tau_i$ but $\sigma_j' \neq \sigma_i$ (otherwise this would not be a winning forgery for I). Consequently $(m_o^j, m_e^j) = (M_o^i, M_e^i)$, but $\sigma_j' \neq \sigma_i$. Also, if $E_{2,1}$ occurs, then F forges. This follows from the fact that F never queried its tagging oracle with (m_n^j, m_t^j) or, if it did, the response was not τ_j'. Lastly, if $E_{2,2}$ occurs, then C wins its game. This is because we know that there is some index i such that $(m_n^j, m_t^j) = (M_n^i, M_t^i)$ but $(m_p^j, m_e^j) \neq (M_p^i, M_e^i)$ (the latter comes from event E_2). Together, this means that the probability that I wins is upper bounded by the sum of the probabilities that C, F, and S win their respective games. The theorem follows for Type 1 AEAD schemes.

We now consider the other types of AEAD schemes. For Type 2, we partition E as follows:

E : I wins

E_1 : E occurs and $(m_p^j, m_e^j, \tau'_j) \in \{\ (M_p^i, M_e^i, \tau_i)\ :\ 1 \leq i \leq k\ \}$

$E_{1,1}$: E_1 occurs and there does not exist i
such that $(m_p^j, \sigma'_j, \tau'_j) = (M_p^i, \sigma_i, \tau_i)$ // S wins

$E_{1,2}$: E_1 occurs and there exists i
such that $(m_p^j, \sigma'_j, \tau'_j) = (M_p^i, \sigma_i, \tau_i)$ // C wins

E_2 : E occurs and $(m_p^j, m_e^j, \tau'_j) \notin \{\ (M_p^i, M_e^i, \tau_i)\ :\ 1 \leq i \leq k\ \}$

$E_{2,1}$: E_2 occurs and $(m_n^j, m_t^j, \tau'_j) \notin \{\ (M_n^i, M_t^i, \tau_i)\ :\ 1 \leq i \leq k\ \}$ // F wins

$E_{2,2}$: E_2 occurs and $(m_n^j, m_t^j, \tau'_j) \in \{\ (M_n^i, M_t^i, \tau_i)\ :\ 1 \leq i \leq k\ \}$ // C wins

This partitioning of event E is the same as with Type 1 except that we further partition event E_1. If event $E_{1,1}$ occurs then S wins (since $(m_o^j, m_e^j) = (M_o^i, M_e^i)$ for some index i but $\sigma'_j \neq \sigma_i$). In the case of $E_{1,2}$, in order for I's j-th decryption query to be considered a forgery, it must be a replayed packet. The first would have been accepted (by the consistency requirements on AEAD schemes). This means that DecodeB failed to return all $\perp$s in response to its second query with m_p^j, m_e^j, allowing C to win.

For Type 3 we partition E as with Type 2. As with Type 2, when $E_{1,2}$ occurs C will win its game (although C's game with Type 3 encoding schemes is different than its game with Type 2 encoding schemes).

For Type 4 we partition E as follows:

E : I wins

E_1 : E occurs and $(m_n^j, m_t^j) \notin \{(M_n^1, M_t^1), \ldots, (M_n^k, M_t^k)\}$ // F wins

E_2 : E occurs and $(m_n^j, m_t^j) \in \{(M_n^1, M_t^1), \ldots, (M_n^k, M_t^k)\}$

$E_{2,1}$: E_2 occurs and either $k < j$ or $(m_p^j, m_e^j) \neq (M_p^j, M_e^j)$ // C wins

$E_{2,2}$: E_2 occurs and $k \geq j$ and $(m_p^j, m_e^j) = (M_p^j, M_e^j)$

$E_{2,2,1}$: $E_{2,2}$ occurs and $\tau'_j \neq \tau_j$ and $(m_n^j, m_t^j) \notin \{(M_n^1, M_t^1), \ldots, (M_n^{j-1}, M_t^{j-1}),$
$(M_n^{j+1}, M_t^{j+1}), \ldots, (M_n^k, M_t^k)\}$ // F wins

$E_{2,2,2}$: $E_{2,2}$ occurs and $\tau'_j \neq \tau_j$ and $(m_n^j, m_t^j) \in \{(M_n^1, M_t^1), \ldots, (M_n^{j-1}, M_t^{j-1}),$
$(M_n^{j+1}, M_t^{j+1}), \ldots, (M_n^k, M_t^k)\}$ // C wins

$E_{2,2,3}$: $E_{2,2}$ occurs and $\tau'_j = \tau_j$ // S wins

If events E_1 or $E_{2,2,1}$ occur then F wins its game; if events $E_{2,1}$ or $E_{2,2,2}$ occur, then C wins its game; if event $E_{2,2,3}$ occurs, then S wins its game. Note that, for $E_{2,2,3}$, we make use of the fact that, as per Construction 4.4.1, once a forgery attempt is detected, the decryption algorithm enters the state $\perp$. This means that, prior to the first forgery attempt, all the decryption-verification queries were in order and, since I's j-th decryption-verification oracle query is a forgery, it must be the case that $\sigma'_j \neq \sigma_j$. (Note that, for Type 4 constructions, if the decryption algorithm didn't enter a halting state we could not guarantee that $\sigma'_j \neq \sigma_j$.) Additionally, by the consistency requirements on the encoding scheme, $m_o^j = M_o^j$.

Let us now consider Type 5. As before, let j denote the index of I's winning decryption-verification-oracle query. Let l be the number of decryption-verification oracle queries (including the j-th query) that succeeded in decrypting (i.e., not returning $(\perp, \perp)$). We partition E as follows:

E : I wins

E_1 : E occurs and $(m_n^j, m_t^j) \notin \{(M_n^1, M_t^1), \ldots, (M_n^k, M_t^k)\}$ // F wins

E_2 : E occurs and $(m_n^j, m_t^j) \in \{(M_n^1, M_t^1), \ldots, (M_n^k, M_t^k)\}$

$E_{2,1}$: E_2 occurs and either $k < l$ or $(m_p^j, m_e^j) \neq (M_p^l, M_e^l)$ // C wins

$E_{2,2}$: E_2 occurs and $k \geq l$ and $(m_p^j, m_e^j) = (M_p^l, M_e^l)$

$E_{2,2,1}$: $E_{2,2}$ occurs and $\tau'_j \neq \tau_l$ and $(m_n^j, m_t^j) \notin \{(M_n^1, M_t^1), \ldots, (M_n^{l-1}, M_t^{l-1}), (M_n^{l+1}, M_t^{l+1}), \ldots, (M_n^k, M_t^k)\}$ // F wins

$E_{2,2,2}$: $E_{2,2}$ occurs and $\tau'_j \neq \tau_l$ and $(m_n^j, m_t^j) \in \{(M_n^1, M_t^1), \ldots, (M_n^{l-1}, M_t^{l-1}), (M_n^{l+1}, M_t^{l+1}), \ldots, (M_n^k, M_t^k)\}$ // C wins

$E_{2,2,3}$: $E_{2,2}$ occurs and $\tau'_j = \tau_l$ // S wins

If events E_1 or $E_{2,2,1}$ occur then F wins its game. Furthermore, if events $E_{2,1}$ or $E_{2,2,2}$ occur, then C wins its game. And if event $E_{2,2,3}$ occurs, then S wins its game. To see that S wins when $E_{2,2,3}$ occurs, we use the consistency requirement on Type 5 encoding

schemes which imply that $m_o^j = M_o^l$. Furthermore, it must be the case that $\sigma'_j \neq \sigma_l$ since otherwise the j-th decryption-verification query would not be a forgery. ∎

4.6 Generalized Encode-then-MtE Security

4.6.1 Privacy

Theorem 4.6.1 below gives our chosen-plaintext privacy result for generalized Encode-then-MtE constructions. Informally, the theorem states that a Type n, $n \in \{1, \ldots, 5\}$, generalized Encode-then-MtE construction will preserve privacy under chosen-plaintext attacks (be PRIV-CPA-secure) if the underlying encryption scheme preserves privacy against chosen-plaintext attacks, i.e., if the underlying encryption scheme is also PRIV-CPA-secure.

Theorem 4.6.1 (Privacy of Generalized Encode-then-MtE Schemes.) Let $\mathcal{SE}$, $\mathcal{MA}$, and $\mathcal{EC}$, respectively, be an encryption, a message authentication, and an MtE encoding scheme. Let $\mathcal{AE}$ be the AEAD scheme associated to them as per Construction 4.4.2. Then, given any adversary S against $\mathcal{AE}$, there exists an adversary A such that

$$\mathbf{Adv}^{\text{priv-cpa}}_{\mathcal{AE}}(S) \;\leq\; \mathbf{Adv}^{\text{priv-cpa}}_{\mathcal{SE}}(A)\,.$$

Furthermore, A uses the same resources as S except that its input to its oracle may be of different lengths than those of S (due to the encoding). If $\mathcal{EC}$ is nonce-respecting-for-encryption (resp., length-based IV-respecting-for-encryption or random-IV-respecting-for-encryption), then A will be nonce-respecting (resp., length-based IV-respecting or random-IV-respecting). ∎

The proof is similar to that of Theorem 4.5 in [10]; we omit details. We remark that the proof relies on the fact that if the encoding algorithm is run using the same random tape, on two pairs of messages $(M_a, M_s), (M_a, N_s)$ such that $|M_s| = |N_s|$, then the resulting values for M_p and M_o will be the same due to the consistency requirements for MtE encoding schemes in Section 4.3.2.

4.6.2 Integrity

We first formalize a new property for generalized Encode-then-MtE AEAD schemes, analogous to the E&M-SP property for generalized Encode-then-E&M AEAD schemes.

Definition 4.6.2 Fix $n \in \{1, \ldots, 5\}$. Let $\mathcal{SE}$, $\mathcal{MA}$, and $\mathcal{EC}$, respectively, be an encryption, a message authentication, and an MtE encoding scheme. Let $\mathcal{AE} = (\overline{\mathcal{K}}, \overline{\mathcal{E}}, \overline{\mathcal{D}})$ be a Type n AEAD scheme associated to them as per Construction 4.4.2. Let A be an adversary with access to an encryption oracle $\overline{\mathcal{E}}_K(\cdot, \cdot)$ and a decryption oracle $\overline{\mathcal{D}}_K(\cdot)$. Let (M_a^i, M_s^i) denote the adversary's i-th encryption oracle query, $(M_p^i, M_o^i, M_n^i, M_e^i, M_t^i)$ denote the encoding of that query, τ_i denote the intermediate tag, and $\langle M_p^i, \sigma_i \rangle$ denote the returned ciphertext. Let $\langle m_p^i, \sigma_i' \rangle$ denote the i-th decryption query (assuming it is parsable), τ_i' denote the intermediate tag, and $m_o^i, m_n^i, m_e^i, m_t^i, m_a^i, m_s^i$ denote the internal values in the decryption process (or $\perp$ if an error occurs during decryption). A "wins" if it makes a decryption query $\langle m_p^j, \sigma_j' \rangle$ such that $(m_o^j, m_e^j, \tau_j') = (M_o^i, M_e^i, \tau_i)$ for some $i \in \{1, \ldots, k\}$ but $\sigma_j' \neq \sigma_i$ (where k is the number of $\overline{\mathcal{E}}_K(\cdot, \cdot)$ oracle queries made by A before A's j-th decryption query). We define the MTE-SP-*advantage* of MTE-SP-*adversary* A as

$$\mathbf{Adv}^{\text{mte-sp}}_{\mathcal{AE}}(A) = \Pr\left[K \xleftarrow{\$} \overline{\mathcal{K}} \ : \ A \text{ "wins"} \right] . \ \blacksquare$$

As in Section 4.5, we present a proposition showing that if the underlying encryption scheme is length preserving, then an adversary cannot win the game described above; we omit the proof.

Proposition 4.6.3 Fix $n \in \{1, \ldots, 5\}$. Let $\mathcal{SE}$, $\mathcal{MA}$, and $\mathcal{EC}$, respectively, be an encryption, a MAC, and a Type n MtE encoding scheme. Let $\mathcal{AE} = (\overline{\mathcal{K}}, \overline{\mathcal{E}}, \overline{\mathcal{D}})$ be a Type n AEAD scheme associated to them as per Construction 4.4.2. Let A be an MTE-SP-adversary. If $\mathcal{SE}$'s encryption operation is length-preserving, then

$$\mathbf{Adv}^{\text{mte-sp}}_{\mathcal{AE}}(A) = 0 . \ \blacksquare$$

We now state our integrity result for generalized Encode-then-MtE constructions. Informally, this theorem states that if a Type n, $n \in \{1, \ldots, 5\}$, generalized Encode-then-MtE construction $\mathcal{AE}$ is built from an encryption scheme $\mathcal{SE}$, a MAC $\mathcal{MA}$, and a Type n MtE encoding scheme $\mathcal{EC}$, and if the latter respects the nonce requirements of $\mathcal{SE}$ and $\mathcal{MA}$, then $\mathcal{AE}$ will be AUTHn-secure if $\mathcal{MA}$ is UF-secure, $\mathcal{EC}$ is MTE-SECn-secure, and $\mathcal{AE}$ has the MTE-SP property specified above. As Proposition 4.6.3 shows, it is easy to construct AEAD schemes that have the MTE-SP property. As with the E&M-SECn security property, there exist natural encoding schemes that satisfy the MTE-SECn security definitions.

Theorem 4.6.4 (Integrity of Generalized Encode-then-MtE Schemes.) Fix $n \in \{1, \ldots, 5\}$. Let $\mathcal{SE}$, $\mathcal{MA}$, and $\mathcal{EC}$, respectively, be an encryption, a message authentication, and an MtE encoding scheme. Let $\mathcal{AE}$ be a Type n AEAD scheme associated to them as per Construction 4.4.2. Then, given any AUTHn-adversary I against $\mathcal{AE}$, there exist adversaries F, C, and S such that

$$\mathbf{Adv}^{\text{auth}n}_{\mathcal{AE}}(I) \leq \mathbf{Adv}^{\text{uf}}_{\mathcal{MA}}(F) + \mathbf{Adv}^{\text{mte-sec}n}_{\mathcal{EC}}(C) + \mathbf{Adv}^{\text{mte-sp}}_{\mathcal{AE}}(S) \ .$$

Furthermore, F, C, and S use the same resources as I except that F's messages to its oracles may be of different lengths than I's queries to its oracles (due to encoding) and C's messages to its decoding oracle may have slightly different lengths than I's decryption queries. If $\mathcal{EC}$ is nonce-respecting-for-MACing, then F will be nonce-respecting. ▮

Proof: The proof is based on the proof of Theorem 4.5.4 for generalized Encode-then-E&M constructions. The partitioning of event E for Type 2 and Type 3 differs slightly from the partitioning we used in the proof of Theorem 4.5.4. The difference is because in the generalized Encode-then-MtE construction, the tag is not sent in the clear. The revised partitioning is as follows:

E : I wins

E_1 : E occurs and $(m_p^j, m_e^j, \tau'_j) \in \{\ (M_p^i, M_e^i, \tau_i)\ :\ 1 \leq i \leq k\ \}$

$E_{1,1}$: E_1 occurs and there does not exist i

such that $(m_p^j, \sigma'_j) = (M_p^i, \sigma_i)$ // S wins

$E_{1,2}$: E_1 occurs and there exists i such that $(m_p^j, \sigma'_j) = (M_p^i, \sigma_i)$ // C wins

E_2 : E occurs and $(m_p^j, m_e^j, \tau'_j) \notin \{ (M_p^i, M_e^i, \tau_i) : 1 \leq i \leq k \}$

$E_{2,1}$: E_2 occurs and $(m_n^j, m_t^j, \tau'_j) \notin \{ (M_n^i, M_t^i, \tau_i) : 1 \leq i \leq k \}$ // F wins

$E_{2,2}$: E_2 occurs and $(m_n^j, m_t^j, \tau'_j) \in \{ (M_n^i, M_t^i, \tau_i) : 1 \leq i \leq k \}$ // C wins

The partitioning of E for Type 1, Type 4, and Type 5 is the same as in the proof of Theorem 4.5.4. ∎

4.7 Generalized Encode-then-EtM Security

4.7.1 Privacy

Theorem 4.7.1 below gives our chosen-plaintext privacy result for generalized Encode-then-EtM constructions. Informally, the theorem states that a Type n, $n \in \{1, \ldots, 5\}$, generalized Encode-then-EtM construction will preserve privacy under chosen-plaintext attacks (be PRIV-CPA-secure) if the underlying encryption scheme is PRIV-CPA-secure.

Theorem 4.7.1 (Privacy of Generalized Encode-then-EtM Schemes.) Let $\mathcal{SE}$, $\mathcal{MA}$, and $\mathcal{EC}$, respectively, be an encryption, a message authentication, and an EtM encoding scheme. Let $\mathcal{AE}$ be the AEAD scheme associated to them as per Construction 4.4.3. Then, given any PRIV-CPA adversary S against $\mathcal{AE}$, there exists an adversary A such that

$$\mathbf{Adv}^{\text{priv-cpa}}_{\mathcal{AE}}(S) \leq \mathbf{Adv}^{\text{priv-cpa}}_{\mathcal{SE}}(A) \; .$$

Furthermore, A use the same resources as S except that its inputs to its oracle may be of different lengths than those of S (due to the encoding). If $\mathcal{EC}$ is nonce-respecting-for-encryption (resp., length-based IV-respecting-for-encryption or random-IV-respecting-for-encryption), then A will be nonce-respecting (resp., length-based IV-respecting or random-IV-respecting). ∎

The proof is similar to that of Theorem 4.7 in [10]. We note that the proof relies on the fact that if the encoding algorithm is run using the same random tape, on two pairs of messages $(M_a, M_s), (M_a, N_s)$ such that $|M_s| = |N_s|$, then the resulting values for M_p, M_o, M_n and M_t will be the same per the consistency requirements for EtM encoding schemes specified in Section 4.3.2.

4.7.2 Integrity

Theorem 4.7.2 below gives our integrity result for generalized Encode-then-EtM constructions. Informally, this theorem states that if a Type n, $n \in \{1, \ldots, 5\}$, generalized Encode-then-EtM construction $\mathcal{AE}$ is built from an encryption scheme $\mathcal{SE}$, a MAC $\mathcal{MA}$, and a Type n EtM encoding scheme $\mathcal{EC}$, and if the latter respects the nonce requirements of $\mathcal{SE}$ and $\mathcal{MA}$, then $\mathcal{AE}$ will be AUTHn-secure if $\mathcal{MA}$ is UF-secure and $\mathcal{EC}$ is ETM-SECn-secure. Note that unlike the integrity results for generalized Encode-then-E&M and generalized Encode-then-MtE constructions (Theorems 4.5.4 and 4.6.4), the security of $\mathcal{AE}$ does not depend on an additional property of $\mathcal{AE}$ (like the properties E&M-SP and MTE-SP). As with the E&M-SECn and MTE-SECn security properties for E&M and MtE encoding schemes, there exist natural EtM encoding schemes that satisfy the ETM-SECn security definitions.

Theorem 4.7.2 (Integrity of Generalized Encode-then-EtM Schemes.) Fix $n \in \{1, \ldots, 5\}$. Let $\mathcal{SE}$, $\mathcal{MA}$, and $\mathcal{EC}$, respectively, be an encryption, a message authentication, and an EtM encoding scheme. Let $\mathcal{AE}$ be a Type n AEAD scheme associated to them as per Construction 4.4.3. Then, given any AUTHn adversary I against $\mathcal{AE}$, there exist adversaries F and C such that

$$\mathbf{Adv}^{\text{auth}n}_{\mathcal{AE}}(I) \leq \mathbf{Adv}^{\text{uf}}_{\mathcal{MA}}(F) + \mathbf{Adv}^{\text{etm-sec}n}_{\mathcal{EC}}(C).$$

Furthermore, F and C use the same resources as I except that F's messages to its oracles may be of different lengths than I's queries to its oracles (due to encoding) and C's messages to its decoding oracle may have slightly different lengths than I's decryption queries. If $\mathcal{EC}$ is nonce-respecting-for-MACing, then F will be nonce-respecting. ∎

Proof: The proof is similar to that of Theorem 4.5.4 and Theorem 4.6.4. Let F and C be adversaries that run I and reply to I's oracle queries using their own oracles. Let (M_a^i, M_s^i) denote I's i-th encryption query, let $(M_p^i, M_o^i, M_n^i, M_e^i, M_t^i)$ denote the encoding of that query, and let $\langle M_p^i, \sigma_i, \tau_i \rangle$ denote the returned ciphertext. Let $\langle m_p^i, \sigma_i', \tau_i' \rangle$ denote the i-th decryption-verification query (assuming it is parsable), and m_o^i, m_n^i, m_t^i, m_e^i, m_a^i, m_s^i denote the internal values in the decryption process (or $\perp$ if an error occurs during decryption). Assume that I wins and let j denote the index of its (first) winning decryption-verification query and k denote the number of encryption queries performed at the time I wins. We will prove that either F or C also wins its game.

For the 5 types of AEAD schemes, we consider the following events:

E : I wins

E_1 : E occurs and $(m_n^j, m_t^j, \sigma_j', \tau_j') \notin \{ (M_n^i, M_t^i, \sigma_i, \tau_i) : 1 \leq i \leq k \}$ // F wins

E_2 : E occurs and $(m_n^j, m_t^j, \sigma_j', \tau_j') \in \{ (M_n^i, M_t^i, \sigma_i, \tau_i) : 1 \leq i \leq k \}$ // C wins

Note that if event E occurs then either E_1 or E_2 must occur. Event E_1 implies that the query $(m_n^j, \langle m_t^j, \sigma_j' \rangle, \tau_j')$ is accepted by the MAC verification oracle (otherwise $\langle m_p^j, \sigma_j', \tau_j' \rangle$ would not be a winning query for I) and is such that τ_j' was never returned by the tagging oracle as an answer to query $(m_n^j, \langle m_t^j, \sigma_j' \rangle)$. Therefore, if E_1 occurs then F forges.

Assume that event E_2 occurs. Then there exists an index $i \leq k$ such that $(m_n^j, m_t^j, \sigma_j', \tau_j') = (M_n^i, M_t^i, \sigma_i, \tau_i)$. For Type 1 AEAD schemes, it must be the case that $m_p^j \neq M_p^i$ (otherwise $\langle m_p^j, \sigma_j', \tau_j' \rangle$ would not be a winning query for I). Since $M_p^i \neq m_p^j$ and $(M_n^i, M_t^i) = (m_n^j, m_t^j)$, C wins. For Type 2 and Type 3 AEAD schemes, C also wins if $m_p^j \neq M_p^i$. If $m_p^j = M_p^i$ then for Type 2 AEAD schemes, it must be the case that $\langle m_p^j, \sigma_j', \tau_j' \rangle$ is a replayed packet (otherwise this would not be a winning query for I). Hence (m_p^j, m_e^j) was decoded correctly (i.e., without returning $(\perp, \perp)$) twice. Therefore, C also wins in this case. For Type 3 AEAD schemes, $m_p^j = M_p^i$ implies that $\langle m_p^j, \sigma_j', \tau_j' \rangle$ is a replayed or out-of-order packet (otherwise this would not be a winning query for I). Again, this implies that C wins. For Type 4 AEAD schemes, it must be the case that either $i \neq j$ or $m_p^j \neq M_p^j$ (if $i = j$ and $m_p^j = M_p^j$, then $j \leq k$ and

$\langle m_p^j, \sigma'_j, \tau'_j \rangle = \langle M_p^j, \sigma_j, \tau_j \rangle$, which contradicts the assumption that $\langle m_p^j, \sigma'_j, \tau'_j \rangle$ is a winning query for I). In both of these cases C wins. Finally, for Type 5 AEAD schemes, let l be the number of decryption-verification oracle queries prior to the j-th one that succeeded in decrypting (i.e., did not return $(\perp, \perp)$). Then it must be the case that either $l \neq i - 1$ or $m_p^j \neq M_p^i$ (if $l = i - 1$ and $m_p^j = M_p^i$, then $l + 1 \leq k$ and $\langle m_p^j, \sigma'_j, \tau'_j \rangle = \langle M_p^{l+1}, \sigma_{l+1}, \tau_{l+1} \rangle$, contradicting the assumption that $\langle m_p^j, \sigma'_j, \tau'_j \rangle$ is a winning query for I). In both of these cases C wins. Hence for all AEAD-scheme types, E_2 implies that C wins. ∎

Additional Information

The material in this chapter comes from in-progress work. I was a primary researcher for this work, the full citation of which is currently:

> Tadayoshi Kohno, Adriana Palacio, and John Black. Authenticated-encryption: New notions and constructions. Manuscript, 2006.

5 The CWC Authenticated Encryption Scheme

In addition to creating AEAD schemes from standard encryption and message authentication schemes, there is a push toward producing block cipher-based AEAD schemes [13, 35, 44, 47, 69, 72, 82]. Despite this push, among the previous works there does not exist any block cipher-based AEAD scheme simultaneously having all five of the following properties: provable security, parallelizability, high performance in hardware, high performance in software, and freedom from intellectual property claims. Even though not all applications require all five of the these properties, we believe that many applications will benefit from at least one of these properties. Moreover, applications may need to interoperate with other systems that desire a different subset of properties.

In this chapter we investigate the design of a block cipher-based AEAD scheme having all five of the above properties. Finding an appropriate balance between all five of these properties is, however, not straightforward since natural approaches to addressing some of the properties are actually disadvantageous with respect to other properties. We believe we have overcome these challenges and, in doing so, introduce a new encryption scheme that we call *Carter-Wegman Counter* (*CWC*) mode.

An earlier version of the material in this chapter appears in Fast Software Encryption, volume 3017 of Lecture Notes in Computer Science [50], copyright the IACR.

5.1 Overview

Motivation. One principle motivation for the research in this chapter is IPsec. (IPsec is a suite of cryptographic protocols that the IETF is currently in the process of standardizing.) From a pragmatic perspective, standardization bodies like the IETF, as well as some vendors, prefer patent-free modes over patented modes. For example, the elegant OCB scheme [72] was apparently rejected from the IEEE 802.11 working group because of patent concerns. From a hardware performance perspective, because none of the pre-existing patent-free AEAD schemes are parallelizable, it to impossible to make pre-existing patent-free AEAD schemes run faster than approximately 2 Gbps using conventional ASIC technology and a single processing unit. Nevertheless, future network devices may need to run at 10 Gbps.

CWC. The AEAD scheme that we propose in this chapter has all five of the properties that we mention in the introduction. First, CWC is provably secure. Moreover, our provable security-based analyses helped guide our research and helped us reject other schemes with similar performance properties but with slightly worse provable security bounds. CWC is also parallelizable, which means that we can make CWC run at 10 Gbps when using conventional ASIC technology and AES as the underlying block cipher. One can also implement CWC efficiently in software. Our implementation of CWC using AES runs at about the same speed as the other patent-free modes on 32-bit architectures (Section 5.6); we anticipate significant performance gains on 32-bit CPUs when using more sophisticated implementation techniques, and we also see significantly better performance on 64-bit architectures. (Patented schemes like OCB are still capable of running faster than CWC in software.)

Like the other two pre-existing unpatented block cipher-based AEAD schemes, CCM [82] and EAX [13], CWC avoids patents by using two inter-related but mostly independent modules: one module to “encrypt” the data and one module to “authenticate” the data. Adopting the terminology used in [13], it is because of the two-module structure that we call CWC a *conventional* block cipher-based AEAD scheme. Although

CWC uses two modules, one can implement it efficiently in a single pass. By using the conventional approach, CCM, EAX, and CWC are very much like composition-based AEAD schemes built from existing encryption schemes and MACs. Unlike composition-based AEAD schemes, however, by designing CWC directly from a block cipher, we eliminate redundant steps and fine-tune CWC for efficiency in both hardware and software. For example, we use only one block cipher key, which saves expensive memory access in hardware.

The encryption core of CWC is based on counter (CTR) mode encryption, which is well-known to be efficient and parallelizable. For authentication, we base our design on the Carter-Wegman [81] universal hash function approach for message authentication. Part of the design challenge is to choose an appropriate universal hash function, with appropriate parameters. Since one can parallelize polynomial evaluation (if the polynomial is in x, one can split the polynomial into i interleaved polynomials in x^i), we choose to use a universal hash function consisting of evaluating a polynomial modulo the prime $2^{127} - 1$. Our hash function is similar to Bernstein's hash127 [17] except that Bernstein's hash function is optimized for software performance at the expense of hardware performance. To address this issue, we use larger coefficients than hash127.

Notation. In our research we first created a general approach for combining CTR mode encryption with a universal hash function in order to provide authenticated encryption. We refer to this general approach as CWC (no change in font), and we use CWC-BC to refer to a CWC instantiation with a 128-bit block cipher BC as the underlying block cipher and with the universal hash function summarized above. We use CWC as shorthand for CWC-BC and use CWC-AES to mean CWC-BC with AES [28] as the underlying block cipher. There are other possible instantiations of the general CWC approach, e.g., for legacy 64-bit block ciphers. Since we are targeting new applications, and since a mode using a 128-bit block cipher cannot interoperate with a mode using a 64-bit block cipher, we focus this paper only on our 128-bit CWC instantiation.

Table 5.1 Software performance in clocks per byte for CWC-AES, CCM-AES, and EAX-AES on a Pentium III. Values are averaged over 50 000 samples.

	Linux/gcc-3.2.2					Windows 2000/Visual Studio 6.0				
	Payload message lengths (bytes)					Payload message lengths (bytes)				
Mode	128	256	512	2048	8192	128	256	512	2048	8192
CWC-AES	105.5	88.4	78.9	72.2	70.5	84.7	70.2	62.2	56.5	55.0
CCM-AES	97.9	87.1	82.0	78.0	77.1	64.8	56.7	52.5	49.5	48.7
EAX-AES	114.1	94.9	86.1	79.1	77.5	75.2	61.8	55.3	50.4	49.1

Performance. Let (A, M) be some input to the CWC encryption algorithm, where A is the associated data and M is the payload data. The CWC encryption algorithm derives a universal hash subkey from the block cipher key. Assuming that the universal hash subkey is maintained across invocations, encrypting (A, M) takes $\lceil |M|/128 \rceil + 2$ block cipher invocations. The polynomial used in CWC's universal hashing step will have degree $d = \lceil |A|/96 \rceil + \lceil |M|/96 \rceil$. There are several ways to evaluate this polynomial; details in Section 5.6. As noted above, one can evaluate this polynomial in parallel. Serially, assuming no precomputation, one can evaluate this polynomial using d 127x127-bit multiplies. As another example, assuming n precomputed powers of the hash subkey, which are cheap to maintain in software for reasonable n, we could evaluate the polynomial using $d - m$ 96x127-bit multiplies and m 127x127-bit multiplies, where $m = \lceil (d+1)/n \rceil - 1$.

In hardware using conventional ASIC technology at 0.13 micron, it takes approximately 300 Kgates to reach 10 Gbps throughput for CWC-AES. This is approximately twice as much as OCB, but avoids IP negotiation overhead and royalty payments to three parties. Table 5.1 relates the software performance, on a Pentium III, of CWC-AES to the two other pre-existing patent-free AEAD modes CCM and EAX; the patented modes such as OCB are not included in this table, but are about twice as fast as the times given for the patent-free modes. The implementations used to compute Table 5.1 were written in C by Brian Gladman [34] and all use 128-bit AES keys; the current CWC-AES implementation does not use the above-mentioned precomputation approach for eval-

uating the polynomial. Table 5.1 shows that the current implementations of the three modes have comparable performance in software, the relative "best" depending on the OS/compiler and the length of the message. Using the above-mentioned precomputation approach and switching to assembly, we anticipate reducing the cost of CWC's universal hashing step to approximately 8 cpb, thereby significantly improving the performance of CWC-AES in software compared to CCM-AES and EAX-AES (since the authentication portions of CCM-AES and EAX-AES are limited by the speed of AES but the authentication portion of CWC-AES is limited by the speed of the universal hash function). For comparison, Bernstein's related hash127, which also evaluates a polynomial modulo $2^{127} - 1$ but whose specific structure makes it less attractive in hardware, runs approximately 4 cpb on a Pentium III when written in assembly and using the precomputation approach. On 64-bit G5s, our initial implementation of the hash function runs at approximately 6 cpb, thus suggesting that CWC-AES is also attractive on 64-bit architectures (when running the G5 in 32-bit mode, our implementation runs at approximately 15 cpb).

We do not claim that CWC-AES is efficient on low-end CPUs such as 8-bit smartcards. However, our goal was not to develop an AEAD scheme for such low-end processors.

The patent issue. The patent issue is a peculiar one. While it may seem odd to let patents influence research, we note that doing so is also not uncommon in some sciences. We view this line of research as discovering the most appropriate solution given real-world constraints.

Additional related works. CWC is similar to a combination of McGrew's UST [59] and TMMH [58], where one of the main advantages of CWC over UST+TMMH is CWC's small key size, which, as the author of UST and TMMH notes, can be a bottleneck for UST+TMMH in hardware at high speeds. The integrity portion of CWC builds on top of the Carter-Wegman universal hashing approach to message authentication [81]. The specific hash function CWC uses is similar to Bernstein's hash127 [17],

but is better suited for hardware. Shoup [74] and Nevelsteen and Preneel [64] also researched software optimizations for universal hash functions. Following the original publication of the material in this chapter, Bernstein introduced a new, efficient MAC that uses polynomial evaluation module $2^{130} - 5$ [18]. Rogaway and Wagner released a critique of CCM [73]. For each issue raised in [73], we find that we have addressed the issue (e.g., we designed CWC to be on-line) or we disagree with the issue (e.g., we feel that it is sufficient for new modes of operation to handle arbitrary octet-length, as opposed to arbitrary bit-length, messages; we stress, however, that, if desired, it is easy to modify CWC to handle arbitrary bit-length messages, see Section 5.5).

CWC recently served as the starting point for GCM [60], a new conventional AEAD scheme also having all five of the target properties that we mentioned at the beginning of this chapter. Unlike our design, however, GCM offers weaker security than our construction under some scenarios. In particular, Ferguson [33] describes attacks against GCM when configured to produce small authentication tags. We designed CWC specifically to avoid attack scenarios like the one Ferguson exploits (Section 5.5).

5.2 Preliminaries

Authenticated encryption schemes with associated data. In this chapter we use Rogaway's notion of an AEAD scheme [69]. Recall that we based our definition of a Type 1 AEAD scheme in Chapter 4 on Rogaway's notion except that in Chapter 4 we do not expose a nonce to the caller, we allow the encryption algorithm to be randomized and stateful and the decryption algorithm to be stateful, and we use a left-or-right indistinguishability definition for privacy. As we define it in this chapter, an AEAD scheme $\mathcal{SE} = (\mathcal{K}_e, \mathcal{E}, \mathcal{D})$ consists of three algorithms and is defined over some key space $\mathsf{KeySp}_{\mathcal{SE}}$, some nonce space $\mathsf{NonceSp}_{\mathcal{SE}} = \{0,1\}^n$, n a positive integer, some associated data space $\mathsf{AdSp}_{\mathcal{SE}} \subseteq \{0,1\}^*$, and some payload message space $\mathsf{MsgSp}_{\mathcal{SE}} \subseteq \{0,1\}^*$. We require that membership in $\mathsf{MsgSp}_{\mathcal{SE}}$ and $\mathsf{AdSp}_{\mathcal{SE}}$ can be efficiently tested and that if M, M' are two strings such that $M \in \mathsf{MsgSp}_{\mathcal{SE}}$ and $|M'| = |M|$, then $M' \in \mathsf{MsgSp}_{\mathcal{SE}}$.

The randomized key generation algorithm $\mathcal{K}_e$ returns a key $K \in \mathsf{KeySp}_{\mathcal{SE}}$; we denote this process as $K \xleftarrow{\$} \mathcal{K}_e$. The deterministic encryption algorithm $\mathcal{E}$ takes as input a key $K \in \mathsf{KeySp}_{\mathcal{SE}}$, a nonce $N \in \mathsf{NonceSp}_{\mathcal{SE}}$, associated data $A \in \mathsf{AdSp}_{\mathcal{SE}}$, and a payload message $M \in \mathsf{MsgSp}_{\mathcal{SE}}$, and returns a ciphertext $C \in \{0,1\}^*$; we denote this process as $C \leftarrow \mathcal{E}_K^{N,A}(M)$ or $C \leftarrow \mathcal{E}_K(N, A, M)$. The deterministic decryption algorithm $\mathcal{D}$ takes as input a key $K \in \mathsf{KeySp}_{\mathcal{SE}}$, a nonce $N \in \mathsf{NonceSp}_{\mathcal{SE}}$, a header $A \in \mathsf{AdSp}_{\mathcal{SE}}$, and a string $C \in \{0,1\}^*$ and outputs a message $M \in \mathsf{MsgSp}_{\mathcal{SE}}$ or the special symbol $\perp$ on error; we denote this process as $M \leftarrow \mathcal{D}_K^{N,A}(C)$. We require that $\mathcal{D}_K^{N,A}(\mathcal{E}_K^{N,A}(M)) = M$ for all $K \in \mathsf{KeySp}_{\mathcal{SE}}$, $N \in \mathsf{NonceSp}_{\mathcal{SE}}$, $A \in \mathsf{AdSp}_{\mathcal{SE}}$, and $M \in \mathsf{MsgSp}_{\mathcal{SE}}$. Let $l(\cdot)$ denote the *length function* of $\mathcal{SE}$; i.e., for all keys K, nonces N, headers A, and messages M, $|\mathcal{E}_K^{N,A}(M)| = l(|M|)$.

Under the correct usage of an AEAD scheme, after a random key is selected, the application should never invoke the encryption algorithm twice with the same nonce value until a new key is randomly selected. In order to ensure that a nonce does not repeat, implementations typically use nonces that contain counters. We use the notion of a nonce, rather than simply a counter, because the notion of a nonce is more general and allows developers the freedom to structure the nonces as they desire.

Block ciphers. We define the notion of a block cipher in Section 2.3. For the purposes of this chapter, we restrict ourselves to block ciphers $E\colon\ \mathcal{K} \times \mathcal{D} \to \mathcal{D}$ where $\mathcal{K} = \{0,1\}^k$ and $\mathcal{D} = \{0,1\}^L$, for some positive integers k and L, rather than arbitrary sets $\mathcal{K}$ and $\mathcal{D}$. We use $f \xleftarrow{\$} E$ as short hand for $K \xleftarrow{\$} \{0,1\}^k\ ;\ f \leftarrow E_K$. We call k the key length of E and we call L the block length.

We use the same definition of pseudorandomness from [6, 56] summarized in Section 2.3. As alternative notation for the definition of pseudorandomness, if $E\colon\ \{0,1\}^k \times \{0,1\}^L \to \{0,1\}^L$ and A is an adversary with access to an oracle and that returns a bit, then we define $\mathbf{Adv}_F^{\text{prp}}(A)$ as

$$\mathbf{Adv}_E^{\text{prp}}(A) = \Pr\left[\, f \xleftarrow{\$} E \ :\ A^{f(\cdot)} = 1 \,\right] - \Pr\left[\, g \xleftarrow{\$} \text{Perm}[L] \ :\ A^{g(\cdot)} = 1 \,\right] .$$

As before, $\mathbf{Adv}_F^{\text{prp}}(A)$ denotes the PRP-advantage of A in distinguishing a random in-

stance of E from a random permutation on L-bit strings.

5.3 Specification

Let $\mathsf{BC}: \{0,1\}^{\mathsf{kl}} \times \{0,1\}^{128} \to \{0,1\}^{128}$ be a 128-bit block cipher. Let $\mathsf{tl} \leq 128$ is the desired tag length in bits. Then the CWC mode of operation using BC with tag length tl, $\mathsf{CWC\text{-}BC\text{-}tl} = (\mathcal{K}, \mathsf{CWC\text{-}ENC}, \mathsf{CWC\text{-}DEC})$, is defined as follows. The message spaces are:

$$\begin{aligned}
\mathsf{MsgSp}_{\mathsf{CWC\text{-}BC\text{-}tl}} &= \{\, x \in (\{0,1\}^8)^* \;:\; |x| \leq \mathsf{MaxMsgLen} \,\} \\
\mathsf{AdSp}_{\mathsf{CWC\text{-}BC\text{-}tl}} &= \{\, x \in (\{0,1\}^8)^* \;:\; |x| \leq \mathsf{MaxAdLen} \,\} \\
\mathsf{KeySp}_{\mathsf{CWC\text{-}BC\text{-}tl}} &= \{0,1\}^{\mathsf{kl}} \\
\mathsf{NonceSp}_{\mathsf{CWC\text{-}BC\text{-}tl}} &= \{0,1\}^{88}
\end{aligned}$$

where $\mathsf{MaxMsgLen}$ and $\mathsf{MaxAdLen}$ are both $128 \cdot (2^{32} - 1)$. That is, the payload and associated data spaces for $\mathsf{CWC\text{-}BC\text{-}tl}$ consist of all strings of octets that are at most $2^{32} - 1$ blocks long.

The $\mathsf{CWC\text{-}BC\text{-}tl}$ key generation, encryption, and decryption algorithms are defined as follows:

Algorithm $\mathcal{K}$

$K \xleftarrow{\$} \{0,1\}^{\mathsf{kl}}$

Return K

Algorithm $\mathsf{CWC\text{-}ENC}_K(N, A, M)$

$\sigma \leftarrow \mathsf{CWC\text{-}CTR}_K(N, M)$

$\tau \leftarrow \mathsf{CWC\text{-}MAC}_K(N, A, \sigma)$

Return $\sigma \| \tau$

Algorithm $\mathsf{CWC\text{-}DEC}_K(N, A, C)$

If $|C| < \mathsf{tl}$ then return $\perp$

Parse C as $\sigma \| \tau$ where $|\tau| = \mathsf{tl}$

If $A \notin \mathsf{AdSp}_{\mathsf{CWC\text{-}BC\text{-}tl}}$ or $\sigma \notin \mathsf{MsgSp}_{\mathsf{CWC\text{-}BC\text{-}tl}}$ then return $\perp$

If $\tau \neq \mathsf{CWC\text{-}MAC}_K(N, A, \sigma)$ then return $\perp$

$M \leftarrow \mathsf{CWC\text{-}CTR}_K(N, \sigma)$

Return M

The algorithms CWC-CTR, CWC-MAC, CWC-HASH are defined below. The CWC-CTR algorithm handles generating the encryption and decryption keystreams, CWC-MAC handles the generation of an authentication tag, and uses CWC-HASH as the underlying universal hash function.

Algorithm $\mathsf{CWC\text{-}CTR}_K(N, M)$

$\alpha \leftarrow \lceil |M|/128 \rceil$

For $i = 1$ to α do $s_i \leftarrow \mathsf{BC}_K(10^7 \| N \| \langle i \rangle_{32})$

$x \leftarrow$ first $|M|$ bits of $s_1 \| s_2 \| \cdots \| s_\alpha$

$\sigma \leftarrow x \oplus M$

Return σ

Algorithm $\mathsf{CWC\text{-}MAC}_K(N, A, \sigma)$

$R \leftarrow \mathsf{BC}_K(\mathsf{CWC\text{-}HASH}_K(A, \sigma))$

$\tau \leftarrow \mathsf{BC}_K(10^7 \| N \| 0^{32}) \oplus R$

Return first tl bits of τ

Algorithm $\mathsf{CWC\text{-}HASH}_K(A, \sigma)$

$Z \leftarrow$ last 127 bits of $\mathsf{BC}_K(110^{126})$

$K_h \leftarrow \mathsf{toint}(Z)$

$l \leftarrow$ min integer such that 96 divides $|A \| 0^l|$

$l' \leftarrow$ min integer such that 96 divides $|\sigma \| 0^{l'}|$

$X \leftarrow A \| 0^l \| \sigma \| 0^{l'}$; $\beta \leftarrow |X|/96$

Break X into chunks $X_1, X_2, \ldots, X_\beta$

For $i = 1$ to β do $Y_i \leftarrow \mathsf{toint}(X_i)$

$l_\sigma \leftarrow |\sigma|/8 \,;\; l_A \leftarrow |A|/8$

$Y_{\beta+1} \leftarrow 2^{64} \cdot l_A + l_\sigma$

$R \leftarrow Y_1 K_h^\beta + \cdots + Y_\beta K_h + Y_{\beta+1} \bmod 2^{127} - 1$

Return $\langle R \rangle_{128}$

5.4 Theorem Statements

The CWC scheme is a provably secure AEAD scheme assuming that the underlying block cipher, e.g., AES, is PRP-secure. As noted in Section 2.3, this assumption is reasonable since most modern block ciphers, including AES, are believed to be PRP-secure. Furthermore, all provably-secure block cipher modes of operation that we are aware of make at least the same assumptions we make, and some modes, such as OCB [72], require the stronger, albeit still reasonable, assumption of super-pseudorandomness. The specific results for CWC appear as Theorem 5.4.1 and Theorem 5.4.2 below.

5.4.1 Privacy

We first show that if BC is a secure block cipher, then CWC-BC-tl will preserve privacy under chosen-plaintext attacks. In this chapter we use the strong definition of indistinguishability for AEAD schemes from [69]. This notion of privacy under chosen-plaintexts attacks is stronger than the conventional left-or-right notion. Let $\mathcal{SE} = (\mathcal{K}_e, \mathcal{E}, \mathcal{D})$ be an AEAD scheme with length function $l(\cdot)$. Let $\$(\cdot,\cdot,\cdot)$ be an oracle that, on input $(N, A, M) \in \mathsf{NonceSp}_{\mathcal{SE}} \times \mathsf{AdSp}_{\mathcal{SE}} \times \mathsf{MsgSp}_{\mathcal{SE}}$, returns a random string of length $l(|M|)$. Let B be an adversary with access to an oracle and that returns a bit. Then

$$\mathbf{Adv}^{\text{priv\$-cpa}}_{\mathcal{SE}}(B) = \Pr\left[\, K \xleftarrow{\$} \mathcal{K}_e \;:\; B^{\mathcal{E}_K(\cdot,\cdot,\cdot)} = 1 \,\right] - \Pr\left[\, B^{\$(\cdot,\cdot,\cdot)} = 1 \,\right]$$

is the PRIV$-CPA-advantage of B in breaking the privacy of $\mathcal{SE}$ under chosen-plaintext attacks; i.e., $\mathbf{Adv}^{\text{priv\$-cpa}}_{\mathcal{SE}}(B)$ is the advantage of B in distinguishing between ciphertexts

from $\mathcal{E}_K(\cdot,\cdot,\cdot)$ and random strings. An adversary B is *nonce-respecting* if it never queries its oracle with the same nonce twice. In the concrete setting [6], a scheme $\mathcal{SE}$ preserves privacy under chosen plaintext attacks (is PRIV$-CPA-secure) if the PRIV$-CPA-advantage of all nonce-respecting adversaries using reasonable resources is small.

Theorem 5.4.1 (Privacy of CWC.) Let CWC-BC-tl be as in Section 5.3. Then given a nonce-respecting PRIV$-CPA adversary A against CWC-BC-tl one can construct a PRP adversary C_A against BC such that if A makes at most q oracle queries totaling at most μ bits of payload message data, then

$$\mathbf{Adv}^{\text{priv\$-cpa}}_{\text{CWC-BC-tl}}(A) \le \mathbf{Adv}^{\text{prp}}_{\text{BC}}(C_A) + \frac{(\mu/128 + 3q + 1)^2}{2^{129}} \ . \tag{5.1}$$

Furthermore, the experiment for C_A takes the same time as the experiment for A and C_A makes at most $\mu/128 + 3q + 1$ oracle queries. ∎

We prove Theorem 5.4.1 in Section 5.7. Let us elaborate on why Theorem 5.4.1 implies that CWC-BC will preserve privacy under chosen-plaintext attacks. Assume BC is a secure block cipher. This means that $\mathbf{Adv}^{\text{prp}}_{\text{BC}}(C)$ must be small for all adversaries C using reasonable resources and, in particular, this means that, for C_A as described in the theorem statement, $\mathbf{Adv}^{\text{prp}}_{\text{BC}}(C_A)$ must be small assuming that A uses reasonable resources. And if $\mathbf{Adv}^{\text{prp}}_{\text{BC}}(C_A)$ is small and μ, q are small, then, because of the above equations, $\mathbf{Adv}^{\text{priv\$-cpa}}_{\text{CWC-BC-tl}}(A)$ must also be small as well. I.e., any adversary A using reasonable resources will only be able to break the privacy of CWC-BC-tl with some small probability.

As a concrete example, let us consider limiting the number of applications of CWC-BC-tl between rekeyings to some reasonable value such as $q = 2^{32}$, and let us limit the total number of payload bits between rekeyings to $\mu = 2^{50}$. Then Equation 5.1 becomes

$$\mathbf{Adv}^{\text{priv\$-cpa}}_{\text{CWC-BC-tl}}(A) \le \mathbf{Adv}^{\text{prp}}_{\text{BC}}(C_A) + \frac{1}{2^{42}}$$

which means that, assuming that the underlying block cipher is a secure PRP, an attacker will not be able to break the privacy of CWC-BC-tl with advantage much greater than 2^{-42}.

5.4.2 Integrity

We now present our results showing that if BC is a secure block cipher, then CWC-BC-tl will protect the authenticity of encapsulated data. We use the strong notion of authenticity for AEAD schemes from [69]. This definition is similar to the definitions of AUTHC integrity from Section 2.6 and AUTH1 integrity from Section 4.1.2, but stated for AEAD schemes as defined in this chapter. Let $\mathcal{SE} = (\mathcal{K}_e, \mathcal{E}, \mathcal{D})$ be an AEAD scheme. Let F be a forging adversary and consider an experiment in which we first pick a random key $K \xleftarrow{\$} \mathcal{K}_e$ and then run F with oracle access to $\mathcal{E}_K(\cdot,\cdot,\cdot)$. We say that F *forges* if F returns a pair (N, A, C) such that $\mathcal{D}_K^{N,A}(C) \neq \perp$ but F did not make a query (N, A, M) to $\mathcal{E}_K(\cdot,\cdot,\cdot)$ that resulted in a response C. Then

$$\mathbf{Adv}^{\text{authc}}_{\mathcal{SE}}(F) = \Pr\left[\, K \xleftarrow{\$} \mathcal{K}_e \;:\; F^{\mathcal{E}_K(\cdot,\cdot,\cdot)} \text{ forges} \,\right]$$

is the AUTHC-*advantage* of F in breaking the integrity/authenticity of $\mathcal{SE}$. Intuitively, the scheme $\mathcal{SE}$ preserves integrity/authenticity if the AUTHC-advantage of all nonce-respecting adversaries using reasonable resources is small.

Theorem 5.4.2 (Integrity/authenticity of CWC.) Let CWC-BC-tl be as specified in Section 5.3. (Recall that BC is a 128-bit block cipher and that the tag length tl is ≤ 128.) Consider a nonce-respecting AUTHC adversary A against CWC-BC-tl. Assume the execution environment allows A to query its oracle with associated data that are at most $n \leq$ MaxAdLen bits long and with messages that are at most $m \leq$ MaxMsgLen bits long. Assume A makes at most $q-1$ oracle queries and the total length of all the payload data (both in these $q-1$ oracle queries and the forgery attempt) is at most μ. Then given A we can construct a PRP adversary C_A against BC such that

$$\mathbf{Adv}^{\text{authc}}_{\text{CWC-BC-tl}}(A) \leq \mathbf{Adv}^{\text{prp}}_{\text{BC}}(C_A) + \frac{(\mu/128 + 3q + 1)^2}{2^{129}} + \frac{n+m}{2^{133}} + \frac{1}{2^{125}} + \frac{1}{2^{\text{tl}}}\ . \quad (5.2)$$

Furthermore, the experiment for C_A takes the same time as the experiment for A and C_A makes at most $\mu/128 + 3q + 1$ oracle queries. ∎

We prove Theorem 5.4.2 in Section 5.7. Let us elaborate on why Theorem 5.4.2 implies that CWC-BC will preserve authenticity. Assume BC is a secure block cipher.

This means that $\mathbf{Adv}^{\text{prp}}_{\mathsf{BC}}(C)$ must be small for all adversaries C using reasonable resources and, in particular, this means that, for C_A as described in the theorem statement, $\mathbf{Adv}^{\text{prp}}_{\mathsf{BC}}(C_A)$ must be small assuming that A uses reasonable resources. And if $\mathbf{Adv}^{\text{prp}}_{\mathsf{BC}}(C_A)$ is small and μ, q, m and n are small, then, because of the above equations, $\mathbf{Adv}^{\text{authc}}_{\mathsf{CWC\text{-}BC\text{-}t|}}(A)$ must also be small as well. I.e., any adversary A using reasonable resources will only be able to break the authenticity of CWC-BC-t| with some small probability.

Let us consider some concrete examples. Let $n = \mathsf{MaxAdLen}$ and $m = \mathsf{MaxMsgLen}$, which is the maximum possible allowed by the CWC-BC construction. Then Equation 5.2 becomes

$$\mathbf{Adv}^{\text{authc}}_{\mathsf{CWC\text{-}BC\text{-}t|}}(A) \leq \mathbf{Adv}^{\text{prp}}_{\mathsf{BC}}(C_A) + \frac{(\mu/128 + 3q + 1)^2}{2^{129}} + \frac{1}{2^{93}} + \frac{1}{2^{\mathsf{t|}}} \ .$$

If we set $q = 2^{32}$ and $\mu = 2^{50}$ as before, and if we take $\mathsf{t|} \geq 43$, then the above equation becomes

$$\mathbf{Adv}^{\text{authc}}_{\mathsf{CWC\text{-}BC\text{-}t|}}(A) \leq \mathbf{Adv}^{\text{prp}}_{\mathsf{BC}}(C_A) + \frac{1}{2^{41}}$$

which means that, assuming that the underlying block cipher is a secure PRP, an attacker will not be able to break the unforgeability of CWC-BC-t| with probability much greater than 2^{-41}.

Chosen-ciphertext privacy. Since the CWC-BC-t| scheme preserves privacy under chosen-plaintext attacks (Theorem 5.4.1) *and* provides integrity (Theorem 5.4.2) assuming that BC is a secure pseudorandom permutation, it also provides privacy under chosen-ciphertext attacks under the same assumption about BC. Recall Sections 2.6, 3.6, and 4.1.2 for a discussion of the relationship between chosen-plaintext privacy, integrity, and chosen-ciphertext privacy for authenticated encryption schemes.

5.5 Design Decisions

Finding an appropriate balance between provable security, hardware efficiency, and software efficiency, while simultaneously avoiding existing intellectual property is-

sues, was one of the principle components of this research project. In this section we discuss how our diverse set of goals affected our design decisions.

The CWC-HASH universal hash function. We chose to simultaneously achieve our parallelizability, hardware, and software goals by basing the authentication portion of CWC on the Carter-Wegman [81] universal hash function approach to message authentication. This is because universal hash functions, and especially the one we created for CWC, can be implemented in multiple ways, thus allowing different platforms and applications to implement CWC-HASH in the way most appropriate for them. For example, hardware implementations will likely parallelize the computation of CWC-HASH by splitting it into multiple polynomials in K_h^i for some i. In more detail, if the polynomial is

$$Y_1 K_h^\beta + Y_2 K_h^{\beta-1} + Y_3 K_h^{\beta-2} + Y_4 K_h^{\beta-3} + \cdots + Y_\beta K_h + Y_{\beta+1} \bmod 2^{127} - 1 \ .$$

then, setting $i = 2$, and $y = K_h^2 \bmod 2^{127} - 1$, and assuming β is odd for illustration purposes, we can rewrite the above polynomial as

$$\Big(Y_1 y^m + Y_3 y^{m-1} + \cdots + Y_\beta\Big)x + \Big(Y_2 y^m + Y_4 y^{m-1} + \cdots + Y_{\beta+1}\Big) \bmod 2^{127} - 1 \ ,$$

After splitting the polynomial, hardware implementations will then likely compute each polynomial using Horner's rule (e.g., the polynomial $aK_h^{2i} + bK_h^i + c$ would be evaluated as $(((a)K_h^i + b)K_h^i) + c$). Software implementations on modern CPUs, for which memory is cheap, will likely precompute a number of powers of K_h and evaluate the CWC-HASH polynomial directly, or almost directly, using a hybrid between a precomputation approach and Horner's rule. We consider a number of possible implementation strategies in more detail in Section 5.6.

CWC-HASH is an instantiation of the classic polynomial universal hash approach to message authentication [81], and is closely related to Bernstein's hash127 [17], which also evaluates a polynomial modulo $2^{127} - 1$. Although hash127 is very fast in software, its structure makes it less suitable for use on high-speed hardware. In particular,

hash127's use of 32-bit coefficients, while great for software implementations with pre-computed powers of K_h, means that hardware implementations using Horner's rule will be "wasting work." Specifically, even with 32-bit coefficients, incorporating each new coefficient using Horner's rule will require a 127x127-bit multiply because the accumulated value will be 127 bits long. By defining the CWC-HASH coefficients to be 96-bits long, we increase the performance of Horner's rule implementations by a factor of three. (We could have gone even further and made the coefficients 126 bits long, but doing so would have required additional complexity to perform bit and byte shifting within the coefficients.) An alternative approach for increasing the performance of a serial implementation of Horner's rule would be to reduce the size of the CWC-HASH subkey K_h to 96 bits. We discuss why we rejected this option in more detail later, but remark here that there are more efficient strategies than Horner's rule for implementing CWC-HASH in software, and that in a parallelized approach the values K_h^i, $i \geq 2$, will most often be full 127-bit values even if K_h is only 96-bits long.

On using a single key. From a security perspective, it would have been perfectly acceptable, and in fact more traditional, to make the CWC-CTR block cipher key and the two CWC-MAC block cipher keys independent. Like others [13, 82], however, we acknowledge that there are several important reasons for sharing keys between the encryption and authentication portions of modes such as CWC. One of the most important reasons is simplicity of key management. Indeed, fetching key material can be a major bottleneck in high-speed hardware, and minimizing key material is thus important. This fact is also why we derive the hash subkey from the block cipher key rather than use an independent hash subkey. We could have defined a mode that derived a number of essentially independent block cipher and hash keys from a single block cipher key, but doing so would either have required more memory or more computation and, because we have proofs that our construction is secure, would have been unnecessary.

Sharing the block cipher key in the way described above and deriving the hash subkey from the block cipher key did, however, mean that we had to be careful with

our proofs of security. To facilitate our proofs, we took extra care in our design to ensure that there would never be a collision in the plaintext inputs to the block cipher between the different usages of the block cipher. For example, by defining CWC-HASH to produce a 127-bit value as output, we know that the first application of BC to CWC-HASH$_K(A, \sigma)$ in CWC-MAC will always have its first bit set to 0. To avoid a collision with the input to the keystream generator, the block cipher inputs in CWC-CTR always have the first two bits set to 10. When using the block cipher to create the hash subkey K_h, the first two bits of the input are set to 11.

On the choice of parameters. Part of this effort involved specifying the appropriate parameters for the CWC encryption mode. Example parameters include the nonce length and the way the nonce is encoded in the input to the block cipher. We chose to fix these parameters for interoperability purposes, but note that our general approach in Section 5.7 does not have theses parameters fixed. We chose to set the nonce length to 88 bits in order to handle future IPsec sequence numbers. And we chose to set the block counter length to 32 bits in order to allow CWC to be used with IPsec jumbograms and other large packets. We also chose to use big-endian byte ordering for consistency purposes and to maintain compatibility with McGrew's ICM Internet-Draft [57] and the IETF, which strongly favors big-endian byte-ordering.

Handling arbitrary bit-length messages. Since we do not believe that many applications will actually require the ability to encrypt arbitrary bit-length messages, we do not define CWC to take arbitrary bit-length messages as input. That said, we did design CWC in such a way that it will be easy to modify the specification to take arbitrary bit-length messages without affecting interoperability with existing implementations when octet-strings are communicated. For example, one could augment the computation of $Y_{\beta+1}$ in CWC-HASH as follows:

$$r_A \leftarrow |A| \bmod 8 \; ; \; r_\sigma \leftarrow |\sigma| \bmod 8 \; ; \; Y_{\beta+1} \leftarrow 2^{120} \cdot r_A + 2^{112} \cdot r_\sigma + 2^{64} \cdot l_A + l_\sigma \; .$$

Of course, a cleaner approach for handling arbitrary bit-length messages would be to compute $l_A \leftarrow |A|$ and $l_\sigma \leftarrow |\sigma|$ in CWC-HASH. We do not define CWC this way because we do not consider it a good trade-off to define a mode for arbitrary bit-length messages at the expense of octet-oriented systems.

64-bit block ciphers. We did not define CWC for use with 64-bit block ciphers because we are targeting future high-speed cryptographic applications. Nevertheless, one can instantiate the general CWC approach in Section 5.7 with 64-bit block ciphers. A 64-bit instantiation may, however, require several undesirable tradeoffs, e.g., in the length of the nonce.

On the length of the hash subkey. As noted earlier, it is possible to use smaller subkeys K_h in CWC-HASH (simply truncate $\mathsf{BC}_K(110^{126})$ appropriately). Recall that we have fixed the block length of BC to 128 bits. Let hkl denote the length of the hash subkey in an altered construction. If $\mathsf{hkl} < 127$, then the upper-bound in Equation 5.2 becomes

$$\mathbf{Adv}^{\text{prp}}_{\mathsf{BC}}(C_A) + \frac{(\mu/128 + 3q + 1)^2}{2^{129}} + \frac{(n+m)/96 + 2}{2^{\mathsf{hkl}}} + \frac{1}{2^{\mathsf{tl}}} \,.$$

Consider an application that sets hkl to 96. If we replace m and n by their maximum possible values, the upper-bound becomes

$$\mathbf{Adv}^{\text{prp}}_{\mathsf{BC}}(C_A) + \frac{(\mu/128 + 3q + 1)^2}{2^{129}} + \frac{1}{2^{62}} + \frac{1}{2^{\mathsf{tl}}} \,.$$

Since 2^{-62} is already very small (and, in fact, dominated by the $(\mu/128 + 3q + 1)^2 \cdot 2^{-129}$ term for some reasonable values of q and μ), from a provable-security perspective, developers would be justified in using 96-bit hash subkeys.

Rather than use shorter hash subkeys, however, our current CWC instantiation in Section 5.3 uses 127-bit hash subkeys. We do so for several reasons. First, in hardware, to obtain maximum speed, one would parallelize the CWC hash function by evaluating, for example, two polynomials in K_h^2 in parallel. As noted before, since K_h^2 would generally not be 96-bits long, there is no performance advantage with using 96-bit subkeys

K_h in this situation. In software, the use of 96-bit hash subkeys could lead to improved performance when evaluating the polynomial using Horner's rule. However, we estimate that the performance of such a construction will be close to the performance of the current construct when not using Horner's rule but using pre-computed powers of K_h. Since we believe that high-performance implementations will not benefit from the use of 96-bit hash subkeys (i.e., the additional 31 key bits come with no or negligible additional cost), we have chosen to fix the length of our hash subkeys to 127 bits.

There may occasionally be reasons to use a CWC variant with hash subkeys even shorter than 96 bits. When these situations arise, one must exercise caution since the use of the shorter hash subkeys could significantly impact security. For example, using a 64-bit hash subkey would increase the upper-bound on the probability of an adversary forging to around 2^{-30}, which may be too large for some applications.

On computing the tag. In CWC the MAC consists of hashing (A, σ), enciphering the hash with the block cipher, and then XORing the result with some keystream (i.e., in the current proposal the tag is $\mathsf{BC}_K(10^7 \| N \| 0^{32}) \oplus \mathsf{BC}_K(\mathsf{CWC\text{-}HASH}_K(A, \sigma))$).

Instead of the two block cipher applications, one could use $\mathsf{BC}_K(h'_K(N, A, \sigma))$ as the tag, where h' is a modified version of CWC-HASH designed to hash 3-tuples instead of pairs of strings (this is important because the nonce must also be authenticated). The main disadvantage of this approach is that it would change the upper-bound in Equation 5.2 to

$$\mathbf{Adv}^{\text{prp}}_{\mathsf{BC}}(C_A) + \frac{(\mu/128 + 3q + 1)^2}{2^{129}} + q^2 \cdot \left(\frac{n+m}{2^{133}} + \frac{1}{2^{125}} \right) + \frac{1}{2^{\mathsf{tl}}}$$

(note the new q^2 term). If we set $n = \mathsf{MaxAdLen}$, $m = \mathsf{MaxMsgLen}$, $q = 2^{32}$, and $\mu = 2^{50}$, then for any $\mathsf{tl} \geq 29$, we get that the advantage of an adversary in breaking the unforgeability of this modified CWC variant is upper-bounded by 2^{-27}, which, although not extremely large, is worse than the upper-bound of 2^{-41} we get using Equation 5.2. Even if n and m are at most one million blocks long, we see that the integrity upper-bound for the altered CWC construction is worse than the upper-bound for the CWC construction we present in Section 5.3. More generally, this means that for reasonable

values of n, m, q, μ, the insecurity upper-bounds of this alternative will be worse than the insecurity upper-bounds of the CWC mode described in Section 5.3. Furthermore, the upper-bound would be even worse if one keys the hash function with shorter keys, which may happen in some situations.

Another possible way to reduce the number of block cipher invocations necessary to compute the MAC would be to take the output of the current hash function and run it through another hash function that is almost-XOR-universal (see Section 5.7 for a description of this property). However, this approach is not attractive because it requires additional key material. In particular, while this approach may save one block cipher operation, in hardware the block cipher operation is actually smaller and simpler than managing the extra key material, given that the hardware already has a block cipher encryptor running at high speed. One could take another block cipher operation to generate the extra key material, but doing so would largely defeat the purpose, except that this block cipher operation could be precomputed or done in parallel.

Another possibility would be to use something like $\mathsf{BC}_K(N) + Y_1 K_h^{\beta+2} + \cdots + Y_\beta K_h^3 + l_A K_h^2 + l_\sigma K_h \bmod 2^{127} - 1$, encoded as a 127-bit string and truncated to tl bits, as the MAC (here $\mathsf{BC}_K(N)$ is interpreted as an integer). Doing so would, however, result in a new integrity upper-bound

$$\mathbf{Adv}^{\mathrm{prp}}_{\mathsf{BC}}(C_A) + \frac{(\mu/128 + 2q + 1)^2 + 4q + 4}{2^{129}} + \frac{(n+m)/96 + 5}{2^{\mathsf{tl}}} \ .$$

If we take n and m to be MaxAdLen and MaxMsgLen, respectively, then the upper-bound becomes

$$\mathbf{Adv}^{\mathrm{prp}}_{\mathsf{BC}}(C_A) + \frac{(\mu/128 + 2q + 1)^2 + 4q + 4}{2^{129}} + \frac{2^{34}}{2^{\mathsf{tl}}} \ .$$

Compared to Equation 5.2, we see the presence of a $2^{34-\mathsf{tl}}$ term. This means that, in some situations, when using the above upper-bound as a guide for parameter selection, tag lengths must be longer than one might expect. For example, if $\mathsf{tl} = 32$, then the above equation would upper-bound the advantage of an adversary against this modified construction as 1. This means that 32-bit tags should not be used with this modified construction when authenticating long messages. While one might consider this more

of a "certificational" problem than a real problem, we view this property as undesirable. (The attack that Ferguson [33] recently discovered against GCM with small tags exploits the fact that GCM operates as per this CWC alternative.)

EAX2. Motivated by EAX2 [13], one possible alternative to CWC might be to use $\mathsf{BC}_K(1110^5\|N)$ both as the value to encrypt R in CWC-MAC and as the initial counter to CTR mode-encrypt M (with the first two bits of the counter always set to 10). Other EAX2-motivated constructions also exist. For example, the tag might be set to $\mathsf{BC}_K(h(X_0\|N)) \oplus \mathsf{BC}_K(h(X_1\|A)) \oplus \mathsf{BC}_K(h(X_2\|\sigma))$, where X_0, X_1, X_2 are strings, none of which is a prefix of the other, and h is a parallelizable universal hash function, like CWC-HASH but hashing only single strings (as opposed to pairs of strings). Compared to CWC, these alternatives have the ability to take longer nonces as input, and, from a functional perspective, can be applied to strings up to 2^{126} blocks long. But we do not view this as a reason to prefer these alternatives over CWC. From a practical perspective, we do not foresee applications requiring nonces longer than 11 octets, or needing to encrypt messages longer than $2^{32} - 1$ blocks. Moreover, from a security perspective, applications should not encrypt too many packets between rekeyings, implying that even 11 octet nonces should be more than sufficient. We do comment, however, that we believe the alternatives discussed in this paragraph are still more attractive than EAX because, like CWC but unlike EAX, these alternatives are parallelizable.

Using existing MACs. We chose not to base the authentication portion of our new mode on XOR-MAC [5] or PMAC [23] because of patent concerns and our software performance requirements; we chose not to base the authentication portion on software-efficient MACs such as HMAC [3] because of our hardware parallelizability requirement.

5.6 Performance

Hardware. Since one of our main goals is to achieve high performance in hardware and, in particular, to provide a solution for future 10 Gbps IPsec (and other) network devices, we focus first on hardware costs. As we noted earlier, it should take approximately 300 Kgates to achieve 10 Gbps throughput for CWC-AES when using 0.13 micron CMOS ASIC technology. This estimate, which is applicable to AES with all key lengths, includes four AES counter-mode encryption engines, each running at 200 MHz and requiring about 25Kgates each. In addition, there are two 32x128-bit multiply/accumulate engines, each running at 200 MHz with a latency of four clocks, one each for the even and odd polynomial coefficients. Simply keeping these engines "fed" may be challenging, but that is generally true of any 10 Gbps path. There may be better methods to structure an implementation, depending on the particular ASIC vendor library and technology. Regardless of the implementation strategy, 10 Gbps is achievable because of the inherent parallelism of CWC.

Since OCB is CWC's main competitor for high-speed environments, it is worth comparing CWC with OCB instantiated with AES (we do not compare CWC with CCM and EAX here since the latter two are not parallelizable). We first note that CWC-AES saves some gates because we only have to implement AES encryption in hardware, i.e., we do not need to implement the inverse of the block cipher. However, at 10 Gbps OCB still probably requires only about half the silicon area of CWC-AES. The main question for many hardware designers is thus whether the extra silicon area for CWC-AES costs more than three royalty payments, as well as negotiation costs and overhead. With respect to negotiation costs and royalty payments, we note that despite significant demands, to date the relevant parties have not all offered publicly available IP fee schedules. Given this fact, and given today's silicon costs, we believe that the extra silicon for CWC-AES is probably cheaper overall than the negotiation costs and IP fees required for OCB.

Software. One can also implement CWC-AES efficiently in software. Table 5.1 shows timing information for CWC-AES, as well as CCM-AES and EAX-AES, on a 1.133GHz mobile Pentium III dual-booting RedHat Linux 9 (kernel 2.4.20-8) and Windows 2000 SP2. The numbers in the table are the clocks per byte for different message lengths averaged over 50 000 runs and include the entire time for setting up (e.g., expanding the AES key-schedule) and encrypting. All implementations were in C and written by Brian Gladman [34] and use 128-bit AES keys. The Linux compiler was gcc version 3.2.2; the Windows compiler was Visual Studio 6.0. OCB runs at about twice the speeds given in Table 5.1.

From Table 5.1 we conclude that the three patent-free modes, as currently implemented by Gladman, share similar software performances. The "best" performing one appears to depend on OS/compiler and the length of the message being processed. On Linux, it appears that CWC-AES performs slightly better than EAX-AES for all message lengths that we tested, and better than CCM-AES for the longer messages, whereas Gladman's CCM-AES and EAX-AES implementations slightly outperform his CWC-AES implementation on Windows for all the message lengths that we tested.

Note, however, that all the implementations used to compute Table 5.1 were written in C. Furthermore, the current CWC-AES code does not make use of all of the optimization techniques, and in particular precomputation, that we describe below. By switching to assembly and using the additional optimization techniques, we anticipate the speed for CWC-HASH to drop to better than 8 clocks per byte, whereas the speed for the CBC-MAC portion of CCM-AES and EAX-AES will be limited by the speed of AES (the best reported speed for AES on a Pentium III is 14.1 cpb, due to a proprietary library by Helger Lipmaa; Gladman's free hand-optimized Windows assembly implementation runs at 17.5 cpb [54]). Returning to the speed of CWC-HASH, for reference we note that Bernstein's related hash127 [17] runs around 4 cpb on a Pentium III when written in assembly and using the precomputation approach. Bernstein's hash127 also works by evaluating a polynomial modulo $2^{127}-1$; the main difference is that the coefficients for hash127 are 32 bits long, whereas the coefficients for CWC-HASH are 96 bits

long (recall Section 5.5, which discusses why we use 96-bit coefficients). We also note that the performance of CWC-HASH will increase dramatically on 64-bit architectures with larger multiplies; an initial implementation on a G5 using 64-bit integer operations runs at around 6 cpb (when running the G5 in 32-bit mode, the hash function runs at around 15 cpb). Bernstein agrees that it is possible to significantly improve our initial performance results for CWC-HASH [18], but does not give performance numbers in his paper (rather, he proposes a new hash function that evaluates a polynomial modulo $2^{130} - 5$).

Since the implementation of CWC-HASH is more complicated than the implementation of the CWC-CTR portion of CWC, we devote the rest of this section to discussing CWC-HASH.

Precomputation. As noted in Section 5.5, there are two general approaches to implementing CWC-HASH in software. The first is to use Horner's rule. The second is to evaluate the polynomial directly, which can be faster if one precomputes powers of the hash key K_h at setup time (here the powers of K_h can be viewed as an expanded key-schedule). In particular, as noted in Section 5.5, evaluating the polynomial using Horner's rule requires a 127x127-bit multiply for each coefficient, whereas evaluating the polynomial directly using precomputed powers of K_h requires a 96x127-bit multiply for each coefficient. (We discuss in Section 5.5 why we did not make the hash subkey 96-bits, which could have sped up a serial Horner's rule implementation.) Bernstein observed the advantage with precomputation in the context of hash127 [17].

The above description of the precomputation approach assumed that if the polynomial is $Y_1 K_h^{\gamma-1} + \cdots + Y_{\gamma-1} K_h + Y_\gamma$ (i.e., the polynomial has γ coefficients), then we had precomputed the powers of K_h^i for all $i \in \{1, \ldots, \gamma - 1\}$. The precomputation approach extends naturally to the case where we have precomputed the powers K_h^j, $j \in \{1, \ldots, n\}$, for some $n \leq \gamma - 1$. For simplicity, first assume that we know the polynomial has a multiple of n coefficients. For such a polynomial, one processes the first n coefficients (to get $Y_1 K_h^{n-1} + \ldots + Y_{n-1} K_h + Y_n$), then multiplies the intermediate result

by K_h^n (to get $Y_1K_h^{2n-1} + \ldots + Y_{n-1}K_h^{n+1} + Y_nK_h^n$). After that, one can continue processing data with the same precomputed values (to get $Y_1K_h^{2n-1} + \ldots + Y_{2n-1}K_h + Y_{2n}$), and so on. Note that each chunk of n coefficients takes $(n-1)$ 96x127-bit multiplies, and all but the last chunk takes an additional 127x127-bit multiply. Now assume that the number of coefficients m in the polynomial is not necessarily a multiple of n. If m is known in advance, one could first process $m \bmod n$ coefficients, multiply by K_h^n, then process in n-coefficient chunks as before. Alternately, as long as the end of the message is known n coefficients in advance, one could process n-coefficients chunks, and then finish off the final $m \bmod n$ coefficients using Horner's rule. Or, if the number of coefficients in the polynomial is not known until the final coefficient is reached, one could process the message in n-coefficient chunks and then multiply by a precomputed power of K_h^{-1} once the end of the message hash been reached.

Naturally, precomputation requires extra memory, but that is usually cheap and plentiful in a software-based environment. Using 32-bit multiplies, the precomputation approach requires 12 32-bit multiplies per 96-bit coefficient, as well as 17 adds, all of which may carry. In assembly, most of these carry operations can be implemented for free, or close to free by using a special variant of the add instruction that adds in the operand as well as the value of the carry from the previous add operation. But when implemented in C, they will generally compile to code that requires a conditional branch and an extra addition. An implementation using Horner's rule requires an additional four multiplies and three additions with carry per coefficient, adding about 33% overhead, since the multiplies dominate the additions. A 64-bit platform only requires four multiplies and four adds (which may all carry), no matter the implementation strategy taken, which explains why implementations of CWC-HASH for 64-bit architectures are much faster.

Exploiting the parallelism of some instruction sets. On most 32-bit platforms, it turns out that the integer execution unit is not the fastest way to implement CWC-HASH. Many platforms have multimedia instructions that one can use to speed

up the implementation. As another alternative, Bernstein demonstrates that on some platforms one can use the floating point unit to implement this class of universal hash function more efficiently than one can with the integer unit. This is particularly true on the x86 platform where, in contrast to using the standard registers, two floating point multiples can be started in close proximity without introducing a pipeline stall. That is, the x86 can effectively perform two floating-point operations in parallel. The disadvantage of using floating-point registers is that the operands for the individual multiplies need to be small, so that the operations can be done without loss of precision. On the x86, Bernstein multiplies 24-bit values, allowing the sums of product terms to fit into double precision values with 53 bits of precision without loss of information. Bernstein details many ways to optimize this sort of calculation in [17].

There are only two main differences between the structure of the polynomials of Bernstein's hash127 and CWC-HASH. The first is that Bernstein uses signed coefficients, whereas CWC-HASH uses unsigned coefficients; this should have little impact on efficiency. The other difference is that Bernstein uses 32-bit coefficients, whereas CWC-HASH uses 96-bit coefficients. While both solutions average one multiplication per byte when using integer math, Bernstein's solution requires only .75 additions per byte, whereas CWC-HASH requires 1.42 additions per byte, nearly twice as many. Using 32-bit multiplies to build a 96x127 multiplier (assuming precomputation), CWC-HASH should therefore perform no worse than at half the speed of hash127. When using 24-bit floating point coefficients to build a multiply (without applying any non-obvious optimizations), hash127 requires 12 multiplies and 16 adds per 32-bit word. CWC can get by with 8 multiples per word and 12.67 additions per word. This is because a 96-bit coefficient fits exactly into four 24-bit values, meaning we can use a 6x4 multiply for every three words. With 32-bit coefficients, we need to use two 24-bit values to represent each coefficient, resulting in a single 6x2 multiply that needs to be performed for each word.

Gladman's C implementation of CWC-HASH uses floating point arithmetic, but uses Horner's rule instead of performing precomputation to achieve extra speed. Noth-

ing about the CWC hash indicates that it should run any worse than half the speed of hash127, if implemented in a similar manner, in assembly, and using the floating point registers and precomputation. This upper-bound paints an encouraging picture for CWC performance, because hash127 on a Pentium III runs around 4 cpb when implemented in assembly and using the floating point registers and precomputation. This indicates that a well-optimized software version of CWC-HASH should run no slower than 8 cycles per byte on the same machine.

5.7 Security Proofs

Before proving Theorem 5.4.1 and Theorem 5.4.2, we first state results about the general CWC construction; see Lemma 5.7.2 and Lemma 5.7.3 below. We then show how Theorems 5.4.1 and 5.4.2 follow from Lemmas 5.7.2 and 5.7.3. We then prove these two lemmas.

5.7.1 More Definitions

Universal hash functions. A hash function $\mathcal{HF} = (\mathcal{K}_h, \mathcal{H})$ consists of two algorithms and is defined over some key space $\mathsf{KeySp}_{\mathcal{HF}}$, some message space $\mathsf{MsgSp}_{\mathcal{HF}}$, and some hash space $\mathsf{HashSp}_{\mathcal{HF}}$. The randomized key generation algorithm returns a random key $K \in \mathsf{KeySp}_{\mathcal{HF}}$; we denote this as $K \xleftarrow{\$} \mathcal{K}_h$. The deterministic hash algorithm takes a key $K \in \mathsf{KeySp}_{\mathcal{HF}}$ and a message $M \in \mathsf{MsgSp}_{\mathcal{HF}}$ and returns a hash value $h \in \mathsf{HashSp}_{\mathcal{HF}}$; we denote this as $h \leftarrow \mathcal{H}_K(M)$. Let $H \xleftarrow{\$} \mathcal{HF}$ be shorthand for $K \xleftarrow{\$} \mathcal{K}_h$; $H \leftarrow \mathcal{H}_K$.

The hash function $\mathcal{HF}$ is said to be ϵ-*almost universal* (ϵ-AU) if for all distinct $m, m' \in \mathsf{MsgSp}_{\mathcal{HF}}$,

$$\Pr\left[\, H \xleftarrow{\$} \mathcal{HF} \;:\; H(m) = H(m') \,\right] \leq \epsilon \;.$$

The hash function $\mathcal{HF}$ is said to be ϵ-*almost* XOR *universal* (ϵ-AXU) if $\mathsf{HashSp}_{\mathcal{HF}} = \{0,1\}^n$ for some positive integer n and for all distinct $m, m' \in \mathsf{MsgSp}_{\mathcal{HF}}$ and $c \in$

$\{0,1\}^n$,

$$\Pr\left[\, H \xleftarrow{\$} \mathcal{HF} \;:\; H(m) \oplus H(m') = c \,\right] \leq \epsilon\ .$$

Pseudorandom functions. We restate the definition of pseudorandom functions from Section 2.2, using slightly different notation. Let F be a family of functions from D to R. Let A be an adversary with access to an oracle and that returns a bit. Then we define $\mathbf{Adv}^{\text{prf}}_F(A)$ as

$$\mathbf{Adv}^{\text{prf}}_F(A) = \Pr\left[\, f \xleftarrow{\$} F \;:\; A^{f(\cdot)} = 1 \,\right] - \Pr\left[\, g \xleftarrow{\$} \text{Rand}[D,R] \;:\; A^{g(\cdot)} = 1 \,\right]\ .$$

As in Section 2.2, $\mathbf{Adv}^{\text{prf}}_F(A)$ denotes the PRF-advantage of A in distinguishing a random instance of F from a random function from D to R.

Message authentication. We now present the definition of a MAC that we will use for the remainder of this chapter. The definition we give below is like the definition in Section 4.2.2 except that here we restrict ourselves to MACs with stateless and deterministic tagging algorithms. In detail, a nonced message authentication scheme $\mathcal{MA} = (\mathcal{K}_m, \mathcal{T}, \mathcal{V})$ consists of three algorithms and is defined over some key space $\mathsf{KeySp}_{\mathcal{MA}}$, some nonce space $\mathsf{NonceSp}_{\mathcal{MA}}$, some message space $\mathsf{MsgSp}_{\mathcal{MA}}$, and some tag space $\mathsf{TagSp}_{\mathcal{MA}}$. The randomized key generation algorithm returns a key $K \in \mathsf{KeySp}_{\mathcal{MA}}$; we denote this as $K \xleftarrow{\$} \mathcal{K}_m$. The deterministic tagging algorithm $\mathcal{T}$ takes a key $K \in \mathsf{KeySp}_{\mathcal{MA}}$, a nonce $N \in \mathsf{NonceSp}_{\mathcal{MA}}$, and a message $M \in \mathsf{MsgSp}_{\mathcal{MA}}$ and returns a tag $\tau \in \mathsf{TagSp}_{\mathcal{MA}}$; we denote this process as $\tau \leftarrow \mathcal{T}^N_K(M)$ or $\tau \leftarrow \mathcal{T}_K(N,M)$. The deterministic verification algorithm $\mathcal{V}$ takes as input a key $K \in \mathsf{KeySp}_{\mathcal{MA}}$, a nonce $N \in \mathsf{NonceSp}_{\mathcal{MA}}$, a message $M \in \mathsf{MsgSp}_{\mathcal{MA}}$, and a candidate tag $\tau \in \{0,1\}^*$, computes $\tau' = \mathcal{T}^N_K(M)$, and returns accept if $\tau' = \tau$ and returns reject otherwise.

Let F be a forging adversary and consider an experiment in which we first pick a random key $K \xleftarrow{\$} \mathcal{K}_m$ and then run F with oracle access to $\mathcal{T}_K(\cdot,\cdot)$. We say that F *forges* if F returns a triple (N,M,τ) such that $\mathcal{V}^N_K(M,\tau) = \mathsf{accept}$ but F did not make a query (N,M) to $\mathcal{T}_K(\cdot,\cdot)$ that resulted in a response τ. Then

$$\mathbf{Adv}^{\text{uf}}_{\mathcal{MA}}(F) = \Pr\left[\, K \xleftarrow{\$} \mathcal{K}_m \;:\; F^{\mathcal{T}_K(\cdot,\cdot)} \text{ forges} \,\right]$$

denotes the UF-*advantage* of F in breaking the *unforgeability* of $\mathcal{MA}$. An adversary is *nonce-respecting* if it never queries its tagging oracle with the same nonce twice. Intuitively, $\mathcal{MA}$ is unforgeable if the UF-advantage of all nonce-respecting adversaries with reasonable resources is small.

5.7.2 The General CWC Construction

We now describe our generalization of the CWC construction.

Construction 5.7.1 [General CWC.] Let l, L, n, o, t, k be positive integers such that $t \leq L$. (Further restrictions will be placed shortly.) Essentially, l is the length of the input to a PRF (e.g., 128), L is the length of the output from the PRF (e.g., 128), n is the length of the nonce (e.g., 88), o is the length of the offset (e.g., 32), t is the length of the desired tag (e.g., 64 or 128), k is the length of the hash function's keysize (e.g., 127).

Let F be a family of functions from $\{0,1\}^l$ to $\{0,1\}^L$. Let $\mathcal{HF} = (\mathcal{K}_h, \mathcal{H})$ be a family of hash functions with $\mathsf{HashSp}_{\mathcal{HF}} = \{0,1\}^l$ and $\mathsf{KeySp}_{\mathcal{HF}} = \{0,1\}^k$ (and $\mathcal{K}_h$ works by randomly selecting and returning an element from $\{0,1\}^k$ with uniform probability). Let $\mathsf{ctr0}\colon\ \mathbb{Z}_{\lceil k/L \rceil} \to \{0,1\}^l$, $\mathsf{ctr1}\colon\ \{0,1\}^n \times (\mathbb{Z}_{2^o} - \{0\}) \to \{0,1\}^l$ and $\mathsf{ctr2}\colon\ \{0,1\}^n \to \{0,1\}^l$ be efficiently-computable injective functions. If $W = \{\ \mathsf{ctr0}(O)\ :\ O \in \mathbb{Z}_{\lceil k/L \rceil}\ \}$, $X = \{\ \mathsf{ctr1}(N, O)\ :\ N \in \{0,1\}^n, O \in (\mathbb{Z}_{2^o} - \{0\})\ \}$, $Y = \{\mathsf{ctr2}(N) : N \in \{0,1\}^n\}$, and $Z = \{\mathcal{H}_K(M) : K \in \mathsf{KeySp}_{\mathcal{HF}}, M \in \mathsf{MsgSp}_{\mathcal{HF}}\}$, we require that W, X, Y, and Z be pairwise mutually exclusive.

Let $\mathsf{extract}\colon\ \{0,1\}^{\lceil k/L \rceil \cdot L} \to \{0,1\}^k$ be a function that takes as input a $\lceil k/L \rceil \cdot L$-bit string and that outputs a k-bit string. We require that extract always pick the same k bits from the input string and always outputs those bits in the exact same order (e.g., extract returns the first k bits of its input).

Let $\mathcal{SE}[F, \mathcal{HF}] = (\mathcal{K}_e, \mathcal{E}, \mathcal{D})$ be an AEAD scheme built from function family F and hash function $\mathcal{HF}$ and using the above functions extract, ctr0, ctr1, ctr2. We assume that $\mathsf{AdSp}_{\mathcal{SE}[F,\mathcal{HF}]} \times \mathsf{MsgSp}_{\mathcal{SE}[F,\mathcal{HF}]} \subseteq \mathsf{MsgSp}_{\mathcal{HF}}$ and that all messages in $\mathsf{MsgSp}_{\mathcal{SE}[F,\mathcal{HF}]}$ have length at most $L \cdot (2^o - 1)$. Note that the former means that the message space of $\mathcal{HF}$

actually consists of pairs of strings. Let $\mathsf{NonceSp}_{\mathcal{SE}[F,\mathcal{HF}]} = \{0,1\}^n$. Let $\mathcal{SE}[F,\mathcal{HF}]$'s component algorithms be defined as follows:

Algorithm $\mathcal{K}_e$

$f \xleftarrow{\$} F$

$\boxed{K_h \leftarrow \mathsf{extract}(f(\mathsf{ctr0}(0)) \| f(\mathsf{ctr0}(1)) \| \cdots \| f(\mathsf{ctr0}(\lceil k/L \rceil - 1))) \; ; \; H \leftarrow \mathcal{H}_{K_h}}$

Return $\langle f, H \rangle$

Algorithm $\mathcal{E}^{N,A}_{\langle f,H \rangle}(M)$

$\sigma \leftarrow \mathsf{CTR\text{-}MODE}^N_f(M)$

$\boxed{\tau \leftarrow \text{first } t \text{ bits of } (f(\mathsf{ctr2}(N)) \oplus f(H(A,\sigma)))}$

Return $\sigma \| \tau$

Algorithm $\mathcal{D}^{N,A}_{\langle f,H \rangle}(C)$

If $|C| < t$ then return $\perp$

Parse C as $\sigma \| \tau$ where $|\tau| = t$

If $A \notin \mathsf{AdSp}_{\mathcal{SE}[F,\mathcal{HF}]}$ or $\sigma \notin \mathsf{MsgSp}_{\mathcal{SE}[F,\mathcal{HF}]}$ then return $\perp$

$\boxed{\tau' \leftarrow \text{first } t \text{ bits of } (f(\mathsf{ctr2}(N)) \oplus f(H(A,\sigma)))}$

If $\tau \neq \tau'$ return $\perp$

$M \leftarrow \mathsf{CTR\text{-}MODE}^N_f(\sigma)$

Return M

Algorithm $\mathsf{CTR\text{-}MODE}^N_f(X)$

$\alpha \leftarrow \lceil |X|/L \rceil$

For $i = 1$ to α do $Z_i \leftarrow f(\mathsf{ctr1}(N,i))$

$Y \leftarrow (\text{first } |X| \text{ bits of } Z_1 \| Z_2 \| \cdots \| Z_\alpha) \oplus X$

Return Y ∎

Before proceeding we make several observations. Recall that one requirement on the message space for any AEAD scheme is that if it contains any string M, then it contains all strings of length $|M|$. This means that the membership test $\sigma \notin \mathsf{MsgSp}_{\mathcal{SE}[F,\mathcal{HF}]}$ and the application of H to (A,σ) makes sense.

As specified in the definition, $\mathsf{AdSp}_{\mathcal{SE}[F,\mathcal{HF}]} \times \mathsf{MsgSp}_{\mathcal{SE}[F,\mathcal{HF}]} \subseteq \mathsf{MsgSp}_{\mathcal{HF}}$. This means that we $\mathcal{HF}$ is used to hash *pairs* of strings, not just string. This is not a serious restriction since given any hash function that hashes strings it is trivial to construct a hash function that hashes pairs of strings by encoding the pair of strings as a single string in some appropriate manner.

It is also worth commenting on the purpose of ctr0, ctr1, and ctr2. As shown in Construction 5.7.1, these functions are used to derive the inputs to the construction's underlying function f. By requiring that none of the outputs collide (i.e., that the sets W, X, Y, Z in the definition are pairwise mutually exclusive), we ensure that the inputs to f for different purposes never collide. For example, the inputs to f used for counter mode encryption will always be different than the inputs to f when enciphering the output of H.

5.7.3 Security of the General CWC Construction

We now state the following results for all Construction 5.7.1-style AEAD schemes. We shall prove Lemmas 5.7.2 and 5.7.3 in Sections 5.7.5 and 5.7.6, respectively.

Lemma 5.7.2 [Integrity of Construction 5.7.1.] Let $\mathcal{SE}[F,\mathcal{HF}]$ be as in Construction 5.7.1 and let $\mathcal{HF}$ be an ϵ-AU hash function. Then given any nonce-respecting AUTHC adversary A against $\mathcal{SE}[F,\mathcal{HF}]$, we can construct a PRF adversary B_A against F such that

$$\mathbf{Adv}^{\text{authc}}_{\mathcal{SE}[F,\mathcal{HF}]}(A) \leq \mathbf{Adv}^{\text{prf}}_{F}(B_A) + \epsilon + 2^{-t} \ .$$

Furthermore, the experiment for B_A takes the same time as the experiment for A and, if A makes at most $q-1$ oracle queries and a total of at most μ bits of payload data (for both these $q-1$ oracle queries and the forgery attempt), then B_A makes at most $\mu/L + 3q + \lceil k/L \rceil$ oracle queries. ∎

Lemma 5.7.3 [Privacy of Construction 5.7.1.] Let $\mathcal{SE}[F,\mathcal{HF}]$ be as in Construction 5.7.1. Then given a nonce-respecting PRIV$-CPA adversary A against $\mathcal{SE}[F,\mathcal{HF}]$

one can construct a PRF adversary B_A against F such that

$$\mathbf{Adv}^{\text{priv\$-cpa}}_{\mathcal{SE}[F,\mathcal{HF}]}(A) \leq \mathbf{Adv}^{\text{prf}}_{F}(B_A) \ .$$

Furthermore, the experiment for B_A takes the same time as the experiment for A and, if A makes at most q oracle queries totaling at most μ bits of payload data, then B_A makes at most $\mu/L + 3q + \lceil k/L \rceil$ oracle queries. ∎

We interpret these lemmas as follows. Intuitively, the first lemma states that if F is a secure PRF, if $\mathcal{HF}$ is ϵ-AU where ϵ is not too large, and if t is not too small, then $\mathcal{SE}[F, \mathcal{HF}]$ preserves integrity. We comment that most modern block ciphers (e.g., AES) are considered to be secure PRPs (and therefore also secure PRFs up to a birthday term). We also comment that we can construct hash functions $\mathcal{HF}$ with provably small ϵ.

Intuitively, the second lemma states that if F is a secure PRF, then $\mathcal{SE}[F, \mathcal{HF}]$ will preserve privacy.

5.7.4 Proofs of Theorem 5.4.1 and Theorem 5.4.2

The security of the CWC construction from Section 5.3 follows from Lemmas 5.7.2 and 5.7.3 assuming that (1) CWC as described in Section 5.3 is really an instantiation of Construction 5.7.1 and (2) that the hash function used in Section 5.3 is ϵ-AU for some small ϵ. We begin by justifying the second bullet.

Lemma 5.7.4 [CWC-HASH is ϵ-almost universal.] Consider the CWC-BC-tl construction from Section 5.3. Let $\mathcal{HF} = (\mathcal{K}_h, \mathcal{H})$ be the hash function whose key generation algorithm selects a random key K from $\{0,1\}^{127}$ and let $\mathcal{H}_K$ be the CWC-HASH function except that we replace

$$Z \leftarrow \text{last } 127 \text{ bits of } \mathsf{BC}_K(110^{126})$$

with

$$Z \leftarrow K \ .$$

Note that $\mathsf{AdSp}_{\mathsf{CWC\text{-}BC\text{-}tl}} \times \mathsf{MsgSp}_{\mathsf{CWC\text{-}BC\text{-}tl}} \subseteq \mathsf{MsgSp}_{\mathcal{HF}}$; that is, $\mathcal{H}_K$ takes two strings as input. Assume $\mathcal{HF}$ hashes pairs of strings where the first string is always at most $n \leq \mathsf{MaxAdLen}$ bits long and the second string is always at most $m \leq \mathsf{MaxMsgLen}$ bits long. Then $\mathcal{HF}$ is ϵ-almost universal where

$$\epsilon \leq \frac{n+m}{2^{133}} + \frac{1}{2^{125}} \cdot \blacksquare$$

Proof of Lemma 5.7.4: Let (A, σ) and (A', σ') be two distinct inputs to $\mathcal{H}_K$ and let $X = (B_1, \ldots, B_{\beta+1})$ and $Y = (C_1, \ldots, C_{\gamma+1})$ respectively denote their encodings as vectors of 96-bit integers (with $B_{\beta+1}$ and $C_{\gamma+1}$ possibly longer than 96-bits long). Without loss of generality, assume $\beta \leq \gamma$ and let $X' = (B'_1, \ldots, B'_{\gamma+1})$ where $B'_j = 0$ for $j \in \{1, \ldots, \gamma - \beta\}$ and $B'_j = B_{j-\gamma+\beta}$ for $j \in \{\gamma - \beta + 1, \ldots, \gamma + 1\}$ (i.e., prepend $\gamma - \beta$ zero elements to the X vector).

If $(A, \sigma) \neq (A', \sigma')$ then $X' \neq Y$. This follows from the fact that $B'_{\gamma+1}$ and $C_{\gamma+1}$ respectively encode the lengths of A and σ and of A' and σ' and that if $X' = Y$, then the $B'_{\gamma+1} = C_{\gamma+1}$ and $(A, \sigma) = (A', \sigma')$.

Note that $\mathcal{H}_K(A, \sigma) = \mathcal{H}_K(A', \sigma')$ when

$$\begin{aligned}\Big(B'_1 \cdot K_h^{\gamma} + \cdots + B'_{\gamma} \cdot K_h + B'_{\gamma+1}\Big) \\ - \Big(C_1 \cdot K_h^{\gamma} + \cdots + C_{\gamma} \cdot K_h + C_{\gamma+1}\Big) = 0 \bmod 2^{127} - 1 \end{aligned} \tag{5.3}$$

where K_h is the hash key derived from K as specified in CWC-HASH. Since the vectors X' and Y are not equal, $\Big(B'_1 \cdot K_h^{\gamma} + \cdots + B'_{\gamma} \cdot K_h + B'_{\gamma+1}\Big) - \Big(C_1 \cdot K_h^{\gamma} + \cdots + C_{\gamma} \cdot K_h + C_{\gamma+1}\Big)$ is a non-zero polynomial of degree at most γ. Therefore, by the Fundamental Theorem of Algebra, Equation 5.3 has at most γ solution modulo $2^{127} - 1$.

We are interested in the probability, over the 127-bit keys K, that Equation 5.3 is true. We note that all keys K_h modulo $2^{127} - 1$ (except 0) have exactly one ways of occurring and that the 0 key can occur in one additional way (i.e., the all 0 string and the all 1 string). This means that of the 2^{127} possible keys K, at most $\gamma + 1$ can lead to keys K_h such that Equation 5.3 is true.

Finally, note that γ is at most $2 + (n+m)/96$ (the $+2$ comes from the fact that we append 0 bits to A and σ). Consequently

$$\epsilon \leq \frac{\frac{n+m}{96} + 3}{2^{127}} \leq \frac{n+m}{2^{133}} + \frac{1}{2^{125}}$$

as desired. ∎

We now prove Theorem 5.4.1 and Theorem 5.4.2, which are corollaries of Lemmas 5.7.2, 5.7.3, and 5.7.4.

Proof of Theorem 5.4.1 and Theorem 5.4.2: To prove these theorems we must show that the CWC-BC-tl constructions from Section 5.3 are instantiations of Construction 5.7.1. We begin by noting that the block cipher BC in CWC-BC-tl plays the role of F in Construction 5.7.1 and that the hash function CWC-HASH (with the simplified key generation algorithm from Lemma 5.7.4) plays the role of $\mathcal{HF}$ in Construction 5.7.1.

Since BC plays the role of F, we have that $l = L = 128$. Furthermore, as described in Section 5.3, $n = 88$, $o = 32$, $t =$ tl, and $k = 127$. We note that the output the hash function is a 128-bit string whose first bit is always 0. This property, as well as the encodings for the nonce/offsets when encrypting the message and the Carter-Wegman MAC and when generating the hash key, ensure that requisite properties for the interactions between the hash function, ctr0, ctr1, and ctr2.

A direct comparison of the Construction 5.7.1 algorithms and the algorithms from Section 5.3 shows that they are equivalent. CWC-BC-tl is therefore an instantiation of Construction 5.7.1 and the provable security of CWC-BC-tl follows.

Finally, we apply the standard PRF-PRP switching technique [6, 12, 75] in order to model the underlying block cipher as a PRP rather than a PRF in Theorem 5.4.1 and Theorem 5.4.2. ∎

5.7.5 Proof of Lemma 5.7.2

We being by sketching the proof of Lemma 5.7.2. We first show that applying a random function to the output of an ϵ-AU hash function yields an ϵ'-AXU hash function

(Proposition 5.7.6). We then recall the result of Krawczyk [51] that XORing the output of an AXU hash function with a one-time pad yields a secure MAC (Proposition 5.7.8). Such a MAC essentially corresponds to the second and third boxed steps in Construction 5.7.1. (We do not need this final block cipher application if the input to the hash includes the nonce and if we accept a birthday term of the form $q^2\epsilon$.)

We then observe that if we consider a construction like Construction 5.7.1 but with the latter two boxed steps replaced with calls to a secure MAC that tags pairs of strings (A, σ) with nonces N, then that construction would be unforgeable (Proposition 5.7.10). In Proposition 5.7.13 we use the above results to show that $\mathcal{SE}[\text{Rand}[l, L], \mathcal{HF}]$ preserves integrity (where $\mathcal{SE}[\text{Rand}[l, L], \mathcal{HF}]$ is as in Construction 5.7.1). Lemma 5.7.2 follows.

From AU to AXU. Let us begin with the following construction.

Construction 5.7.5 [Building AXU hash functions from AU hash functions.] Let $\mathcal{HF} = (\mathcal{K}_h, \mathcal{H})$ be a hash function and let $\overline{\mathcal{HF}[t]} = (\overline{\mathcal{K}_h}, \overline{\mathcal{H}})$, t a positive integer, be the hash function defined as follows:

$\overline{\mathcal{K}_h}$	$\overline{\mathcal{H}}_{\langle H,e\rangle}(M)$
$H \xleftarrow{\$} \mathcal{HF}$	Return $e(H(M))$
$e \xleftarrow{\$} \text{Rand}[\mathsf{HashSp}_{\mathcal{HF}}, \{0,1\}^t]$	
Return $\langle H, e\rangle$	

Note that $\mathsf{MsgSp}_{\overline{\mathcal{HF}[t]}} = \mathsf{MsgSp}_{\mathcal{HF}}$ and $\mathsf{HashSp}_{\overline{\mathcal{HF}[t]}} = \{0,1\}^t$. ▮

Proposition 5.7.6 Let $\mathcal{HF}$, t, and $\overline{\mathcal{HF}[t]}$ be as in Construction 5.7.5. If $\mathcal{HF}$ is ϵ-AU, then $\overline{\mathcal{HF}[t]}$ is $(\epsilon + 2^{-t})$-AXU. ▮

This result follows from a result in [68, 78] which states that the composition of an ϵ'-AXU hash function, with domain B and range C, with an ϵ-AU hash function, with domain A and range B, is an $(\epsilon + \epsilon')$-AXU hash function with domain A and range C, and the fact that the hash function whose key generation algorithm returns a random function from $\text{Rand}[\mathsf{HashSp}_{\mathcal{HF}}, \{0,1\}^t]$ is 2^{-t}-AXU.

Carter-Wegman MACs. Consider now the following construction.

Construction 5.7.7 [Building MACs from AXU hash functions.] Let $\mathcal{HF} = (\mathcal{K}_h, \mathcal{H})$ be a hash function with hash space $\{0,1\}^t$, t a positive integer. We can construct a nonced message authentication scheme $\mathcal{MA} = (\mathcal{K}_m, \mathcal{T}, \mathcal{V})$ as follows:

$\mathcal{K}_m$
  $H \xleftarrow{\$} \mathcal{HF}$; $g \xleftarrow{\$} \mathsf{Rand}[\mathsf{NonceSp}_{\mathcal{MA}}, \{0,1\}^t]$
  Return $\langle H, g\rangle$

$\mathcal{T}_{\langle H,g\rangle}(N, M)$
  Return $g(N) \oplus H(M)$

$\mathcal{V}_{\langle H,g\rangle}(N, M, \tau)$
  If $g(N) \oplus H(M) = \tau$ then
    return accept
  Else return reject

Note that $\mathsf{MsgSp}_{\mathcal{MA}} = \mathsf{MsgSp}_{\mathcal{HF}}$, $\mathsf{TagSp}_{\mathcal{MA}} = \{0,1\}^t$, and that $\mathsf{NonceSp}_{\mathcal{MA}}$ is arbitrary. ▮

We now state the following result, due to Krawczyk [51].

Proposition 5.7.8 Let $\mathcal{HF}$ and $\mathcal{MA}$ be as in Construction 5.7.7. If $\mathcal{HF}$ is ϵ-AXU, then for all nonce-respecting UF adversaries F attacking $\mathcal{MA}$, $\mathbf{Adv}^{\mathrm{uf}}_{\mathcal{MA}}(F) \leq \epsilon$. ▮

As noted in [51], this proposition follows from the facts that XORing the output of the hash function with $g(N)$ prevents any loss of information (assuming that the adversary is nonce-respecting), that a forgery attempt with a previous nonce is upper-bounded by ϵ, and that a forgery attempt with a new nonce is upper-bounded by $2^{-t} \leq \epsilon$.

Encrypt-then-MAC. Consider the following Encrypt-then-MAC construction.

Construction 5.7.9 [Encrypt-then-MAC.] Let l, L, n, o, t be positive integers. (Further restrictions will be placed shortly.) Essentially, l is the length of the input to a PRF (e.g., 128), L is the length of the output from the PRF (e.g., 128), n is the length of the nonce (e.g., 88), o is the length of the offset (e.g., 32).

Let F be a family of functions from $\{0,1\}^l$ to $\{0,1\}^L$. Let $\mathcal{MA} = (\mathcal{K}_m, \mathcal{T}, \mathcal{V})$ be a message authentication scheme with $\mathsf{NonceSp}_{\mathcal{MA}} = \{0,1\}^n$ and $\mathsf{TagSp}_{\mathcal{MA}} = \{0,1\}^t$.

Let $\mathsf{ctr1}: \{0,1\}^n \times (\mathbb{Z}_{2^o} - \{0\}) \to \{0,1\}^l$ be an efficiently-computable injective function.

Let $\mathcal{SE}[F, \mathcal{MA}] = (\mathcal{K}_e, \mathcal{E}, \mathcal{D})$ be an AEAD scheme built from function family F and message authentication scheme $\mathcal{MA}$ and using the above function $\mathsf{ctr1}$. We assume that $\mathsf{AdSp}_{\mathcal{SE}[F,\mathcal{MA}]} \times \mathsf{MsgSp}_{\mathcal{SE}[F,\mathcal{MA}]} \subseteq \mathsf{MsgSp}_{\mathcal{MA}}$ and that all messages in $\mathsf{MsgSp}_{\mathcal{SE}[F,\mathcal{MA}]}$ have length at most $L \cdot (2^o - 1)$. Note that the former means that the message space of $\mathcal{MA}$ actually consists of pairs of strings. Let $\mathsf{NonceSp}_{\mathcal{SE}[F,\mathcal{MA}]} = \mathsf{NonceSp}_{\mathcal{MA}}$. Let $\mathcal{SE}[F, \mathcal{MA}]$'s component algorithms be defined as follows:

Algorithm $\mathcal{K}_e$

$f \xleftarrow{\$} F$

$\boxed{K \xleftarrow{\$} \mathcal{K}_m}$

Return $\langle f, K \rangle$

Algorithm $\mathcal{E}^{N,A}_{\langle f,K \rangle}(M)$

$\sigma \leftarrow \mathsf{CTR\text{-}MODE}^N_f(M)$

$\boxed{\tau \leftarrow \mathcal{T}^N_K(A, \sigma)}$

Return $\sigma \| \tau$

Algorithm $\mathcal{D}^{N,A}_{\langle f,K \rangle}(C)$

If $|C| < t$ then return $\perp$

Parse C as $\sigma \| \tau$ where $|\tau| = t$

If $A \notin \mathsf{AdSp}_{\mathcal{SE}[F,\mathcal{MA}]}$ or $\sigma \notin \mathsf{MsgSp}_{\mathcal{SE}[F,\mathcal{MA}]}$ then return $\perp$

$\boxed{\tau' \leftarrow \mathcal{T}^N_K(A, \sigma)}$

If $\tau \neq \tau'$ return $\perp$

$M \leftarrow \mathsf{CTR\text{-}MODE}^N_f(\sigma)$

Return M

Algorithm $\mathsf{CTR\text{-}MODE}^N_f(X)$

$\alpha \leftarrow \lceil |X|/L \rceil$

For $i = 1$ to α do

$Z_i \leftarrow f(\mathsf{ctr1}(N, i))$

$Y \leftarrow (\text{first } |X| \text{ bits of } Z_1 \| Z_2 \| \cdots \| Z_\alpha) \oplus X$

Return Y ∎

Proposition 5.7.10 Let $\mathcal{SE}[F, \mathcal{MA}]$ be as in Construction 5.7.9. Then given a nonce-respecting AUTHC adversary B against $\mathcal{SE}[F, \mathcal{MA}]$, we can construct a nonce-respecting forgery adversary D_B against $\mathcal{MA}$ such that

$$\mathbf{Adv}^{\text{authc}}_{\mathcal{SE}[F,\mathcal{MA}]}(B) \leq \mathbf{Adv}^{\text{uf}}_{\mathcal{MA}}(D_B) \ .$$

Furthermore the experiment for D_B uses the same time as the experiment for B and if B makes q encryption oracle queries, then D_B makes q tagging oracle queries. ∎

To prove Proposition 5.7.10, we use the approach in [10] for analyzing Encrypt-then-MAC constructions. The only difference is that we consider MACs that also take nonces as input.

Combining these constructions. Let us now combine these constructions.

Construction 5.7.11 [Combined CWC.] Let l, L, n, o, t, k be positive integers such that $t \leq L$. (Further restrictions will be placed shortly.) Essentially, l is the length of the input to a PRF (e.g., 128), L is the length of the output from the PRF (e.g., 128), n is the length of the nonce (e.g., 88), o is the length of the offset (e.g., 32), t is the length of the desired tag (e.g., 64 or 128), k is the length of the hash function's keysize (e.g., 128).

Let F be a family of functions from $\{0,1\}^l$ to $\{0,1\}^L$. Let $\mathcal{HF} = (\mathcal{K}_h, \mathcal{H})$ be a family of hash functions with $\mathsf{HashSp}_{\mathcal{HF}} = \{0,1\}^l$ and $\mathsf{KeySp}_{\mathcal{HF}} = \{0,1\}^k$ (and $\mathcal{K}_h$ works by randomly selecting and returning an element from $\{0,1\}^k$ with uniform probability). Let $\mathsf{ctr1}: \ \{0,1\}^n \times (\mathbb{Z}_{2^o} - \{0\}) \to \{0,1\}^l$ be an efficiently-computable injective function. Let $\mathsf{extract}: \ \{0,1\}^{\lceil k/L \rceil \cdot L} \to \{0,1\}^k$ be a function that takes as input a $\lceil k/L \rceil \cdot L$-bit string and that outputs a k-bit string. We require that $\mathsf{extract}$ always pick the same k bits from the input string and always outputs those bits in the exact same order (e.g., $\mathsf{extract}$ returns the first k bits of its input).

Let $\mathcal{SE}[F,\mathcal{HF}] = (\mathcal{K}_e, \mathcal{E}, \mathcal{D})$ be an AEAD scheme built from function family F and hash function $\mathcal{HF}$ and using the above functions extract and ctr1. We assume that $\mathsf{AdSp}_{\mathcal{SE}[F,\mathcal{HF}]} \times \mathsf{MsgSp}_{\mathcal{SE}[F,\mathcal{HF}]} \subseteq \mathsf{MsgSp}_{\mathcal{HF}}$ and that all messages in $\mathsf{MsgSp}_{\mathcal{SE}[F,\mathcal{HF}]}$ have length at most $L \cdot (2^o - 1)$. Note that the former means that the message space of $\mathcal{HF}$ actually consists of pairs of strings. Let $\mathsf{NonceSp}_{\mathcal{SE}[F,\mathcal{HF}]} = \{0,1\}^n$. Let $\mathcal{SE}[F,\mathcal{HF}]$'s component algorithms be defined as follows:

Algorithm $\mathcal{K}_e$

$f \xleftarrow{\$} F$

$d \xleftarrow{\$} \mathrm{Rand}[\mathbb{Z}_{\lceil k/L \rceil}, \{0,1\}^L]$

$e \xleftarrow{\$} \mathrm{Rand}[\mathsf{HashSp}_{\mathcal{HF}}, \{0,1\}^t]$

$g \xleftarrow{\$} \mathrm{Rand}[\mathsf{NonceSp}_{\mathcal{SE}[F,\mathcal{HF}]}, \{0,1\}^t]$

$K_h \leftarrow \mathsf{extract}(d(0) \| d(1) \| \cdots \| d(\lceil k/L \rceil - 1))$; $H \leftarrow \mathcal{H}_{K_h}$

Return $\langle f, H, e, g \rangle$

Algorithm $\mathcal{E}^{N,A}_{\langle f,H,e,g \rangle}(M)$

$\sigma \leftarrow \mathsf{CTR\text{-}MODE}^N_f(M)$

$\tau \leftarrow g(N) \oplus e(H(A,\sigma))$

Return $\sigma \| \tau$

Algorithm $\mathcal{D}^{N,A}_{\langle f,H,e,g \rangle}(C)$

If $|C| < t$ then return $\perp$

Parse C as $\sigma \| \tau$ where $|\tau| = t$

If $A \notin \mathsf{AdSp}_{\mathcal{SE}[F,\mathcal{HF}]}$ or $\sigma \notin \mathsf{MsgSp}_{\mathcal{SE}[F,\mathcal{HF}]}$ then return $\perp$

$\tau' \leftarrow g(N) \oplus e(H(A,\sigma))$

If $\tau \neq \tau'$ return $\perp$

$M \leftarrow \mathsf{CTR\text{-}MODE}^N_f(\sigma)$

Return M

Algorithm $\mathsf{CTR\text{-}MODE}^N_f(X)$

$\alpha \leftarrow \lceil |X|/L \rceil$

For $i = 1$ to α do

$Z_i \leftarrow f(\mathsf{ctr1}(N, i))$

$Y \leftarrow (\text{first } |X| \text{ bits of } Z_1 \| Z_2 \| \cdots \| Z_\alpha) \oplus X$

Return Y ▮

Proposition 5.7.12 Let $\mathcal{SE}[F, \mathcal{HF}]$ be as in Construction 5.7.11 and let $\mathcal{HF}$ be an ϵ-AU hash function. Then the advantage of any nonce-respecting AUTHC adversary A in breaking the authenticity of $\mathcal{SE}[F, \mathcal{HF}]$ is upper bounded by

$$\mathbf{Adv}^{\text{authc}}_{\mathcal{SE}[F,\mathcal{HF}]}(A) \leq \epsilon + 2^{-t} \ . \ ▮$$

Proof of Proposition 5.7.12: We first note that the steps $d \xleftarrow{\$} \text{Rand}[\mathbb{Z}_{\lceil k/L \rceil}, \{0,1\}^L]$; $K_h \leftarrow \mathsf{extract}(d(0) \| d(1) \| \cdots \| d(\lceil k/L \rceil - 1))$; $H \leftarrow \mathcal{H}_{K_h}$ is equivalent to the step $H \xleftarrow{\$} \mathcal{HF}$.

Note that $e(H(A, \sigma))$ can be rewritten as $\overline{\mathcal{H}}_{\langle H,e \rangle}(A, \sigma)$ where $\overline{\mathcal{HF}[t]} = (\overline{\mathcal{K}_h}, \overline{\mathcal{H}})$ is composed from $\mathcal{HF}$ per Construction 5.7.5.

Also note that $g(N) \oplus \overline{\mathcal{H}}_{\langle H,e \rangle}(A, \sigma)$ can be replaced with $\mathcal{T}^N_{\langle \overline{\mathcal{H}}_{\langle H,e \rangle}, g \rangle}(A, \sigma)$ where $\mathcal{MA} = (\mathcal{K}_m, \mathcal{T}, \mathcal{V})$ is composed from $\overline{\mathcal{HF}[t]}$ as per Construction 5.7.7.

By Proposition 5.7.10, given A we can construct an adversary B_A against $\mathcal{MA}$ such that

$$\mathbf{Adv}^{\text{authc}}_{\mathcal{SE}[F,\mathcal{HF}]}(A) \leq \mathbf{Adv}^{\text{uf}}_{\mathcal{MA}}(B_A) \ .$$

By Proposition 5.7.8 we know that

$$\mathbf{Adv}^{\text{uf}}_{\mathcal{MA}}(B_A) \leq \epsilon'$$

where ϵ' is $\epsilon + 2^{-t}$ (the latter by Proposition 5.7.6). ▮

Integrity of $\mathcal{SE}[\mathbf{Rand}[l, L], \mathcal{HF}]$. We now consider the integrity of $\mathcal{SE}[\text{Rand}[l, L], \mathcal{HF}]$.

Proposition 5.7.13 Let $\mathcal{SE}[\text{Rand}[l, L], \mathcal{HF}]$ be a AEAD scheme as in Construction 5.7.1. Then for any nonce-respecting AUTHC adversary A against $\mathcal{SE}[\text{Rand}[l, L], \mathcal{HF}]$, we have that

$$\mathbf{Adv}^{\text{authc}}_{\mathcal{SE}[\text{Rand}[l,L],\mathcal{HF}]}(A) \leq \epsilon + 2^{-t} \ . \ \blacksquare$$

Proof of Proposition 5.7.13: Let $\mathcal{SE}'[\text{Rand}[l, L], \mathcal{HF}]$ be as in Construction 5.7.11. Note that $\mathcal{SE}[\text{Rand}[l, L], \mathcal{HF}]$ and $\mathcal{SE}'[\text{Rand}[l, L], \mathcal{HF}]$ are identical except that the former uses only one random function f and $\mathcal{SE}'[\text{Rand}[l, L], \mathcal{HF}]$ uses four random functions (one to generate the hash key, one to CTR-mode encrypt the message, one to encipher the output of the hash function, and one to CTR-mode encrypt the output of the hash function). Furthermore, recall that, for $\mathcal{SE}[\text{Rand}[l, L], \mathcal{HF}]$, there is never a collision in the input to f between the four different uses of f (this was a requirement imposed on $\mathcal{HF}$, ctr0, ctr1, and ctr2). Consequently, the fact that $\mathcal{SE}'[\text{Rand}[l, L], \mathcal{HF}]$ uses four random functions and $\mathcal{SE}[\text{Rand}[l, L], \mathcal{HF}]$ uses one is immaterial. Hence the probability that A forges against $\mathcal{SE}[\text{Rand}[l, L], \mathcal{HF}]$ is the same as the probability that it forges against $\mathcal{SE}'[\text{Rand}[l, L], \mathcal{HF}]$. I.e.,

$$\mathbf{Adv}^{\text{authc}}_{\mathcal{SE}[\text{Rand}[l,L],\mathcal{HF}]}(A) = \mathbf{Adv}^{\text{authc}}_{\mathcal{SE}'[\text{Rand}[l,L],\mathcal{HF}]}(A) \ .$$

By Proposition 5.7.12, we know the latter probability is upper bounded by $\epsilon + 2^{-t}$. $\blacksquare$

Proof of Lemma 5.7.2. We now prove Lemma 5.7.2.

Proof of Lemma 5.7.2: Adversary B_A runs A and replies to A's oracle queries using its oracle f. If A returns a valid forgery, B_A returns 1, otherwise B_A returns 0. This implies that

$$\mathbf{Adv}^{\text{authc}}_{\mathcal{SE}[F,\mathcal{HF}]}(A) = \Pr\left[\ f \xleftarrow{\$} F \ : \ B_A^{f(\cdot)} = 1 \ \right]$$

and

$$\mathbf{Adv}^{\text{authc}}_{\mathcal{SE}[\text{Rand}[l,L],\mathcal{HF}]}(A) = \Pr\left[\ f \xleftarrow{\$} \text{Rand}[l, L] \ : \ B_A^{f(\cdot)} = 1 \ \right] \ .$$

Since

$$\mathbf{Adv}^{\text{authc}}_{\mathcal{SE}[\text{Rand}[l,L],\mathcal{HF}]}(A) \leq \epsilon + 2^{-t}$$

by Proposition 5.7.13, we have

$$\begin{aligned}\mathbf{Adv}^{\text{authc}}_{\mathcal{SE}[F,\mathcal{HF}]}(A) &= \mathbf{Adv}^{\text{authc}}_{\mathcal{SE}[F,\mathcal{HF}]}(A) - \mathbf{Adv}^{\text{authc}}_{\mathcal{SE}[\text{Rand}[l,L],\mathcal{HF}]}(A) \\ &\quad + \mathbf{Adv}^{\text{authc}}_{\mathcal{SE}[\text{Rand}[l,L],\mathcal{HF}]}(A) \\ &\leq \Pr\left[\, f \xleftarrow{\$} F \;:\; B_A^{f(\cdot)} = 1 \,\right] - \Pr\left[\, f \xleftarrow{\$} \text{Rand}[l,L] \;:\; B_A^{f(\cdot)} = 1 \,\right] \\ &\quad + \epsilon + 2^{-t} \\ &= \mathbf{Adv}^{\text{prf}}_{F}(B_A) + \epsilon + 2^{-t}\end{aligned}$$

as desired. ∎

5.7.6 Proof of Lemma 5.7.3

Proof of Lemma 5.7.3: Let B_A be a PRF adversary against F that uses adversary A and that has oracle access to a function $g\colon\ \{0,1\}^l \to \{0,1\}^L$. Adversary B_A runs A and replies to A's encryption oracle queries using its own oracle $g(\cdot)$ for the function f in Construction 5.7.1. Adversary B_A returns the same bit that A returns. Then

$$\Pr\left[\, \langle f, H\rangle \xleftarrow{\$} \mathcal{K}_e \;:\; A^{\mathcal{E}_{\langle f,H\rangle}(\cdot,\cdot,\cdot)} = 1 \,\right] = \Pr\left[\, g \xleftarrow{\$} F \;:\; B_A^{g(\cdot)} = 1 \,\right]$$

since when B_A is given a random instance of F it runs A exactly as if A was given the real encryption oracle. Furthermore

$$\Pr\left[\, A^{\$(\cdot,\cdot,\cdot)} = 1 \,\right] = \Pr\left[\, g \xleftarrow{\$} \text{Rand}[l,L] \;:\; B_A^{g(\cdot)} = 1 \,\right]$$

since B_A replies to all of A's oracle queries with independently selected random strings. Consequently

$$\mathbf{Adv}^{\text{priv\$-cpa}}_{\mathcal{SE}[F,\mathcal{HF}]}(A) \leq \mathbf{Adv}^{\text{prf}}_{F}(B_A)$$

as desired. ∎

Additional Information

An earlier version of the material in this chapter appears in Fast Software Encryption, volume 3017 of Lecture Notes in Computer Science [50], copyright the IACR. I

was a primary researcher for the theoretical results in this paper. The full citation for this work is:

> Tadayoshi Kohno, John Viega, and Doug Whiting. CWC: A high-performance conventional authenticated encryption mode. In Bimal Roy and Willi Meier, editors, *Fast Software Encryption*, volume 3017 of *Lecture Notes in Computer Science*, pages 408–426. Springer-Verlag, February 2004.

6 The WinZip Authenticated Encryption Scheme

WinZip [85] is a popular compression utility for Microsoft Windows computers, the latest version of which is advertised as having "easy-to-use AES encryption to protect your sensitive data" [85]. Because of WinZip's already established large user base, and because of its advertised encryption feature, we anticipate that many current and future users will choose to exercise this encryption option with the hopes of cryptographically protecting their personal data. Additionally, because of WinZip's Microsoft Outlook email plugin [84] and given other comments on WinZip's websites [85, 86], we anticipate that many users will also choose to use WinZip's encryption feature in an attempt to cryptographically protect the contents of their email attachments and other shared data.

Unfortunately, WinZip's new encryption scheme, called "Advanced Encryption-2" or AE-2 [83] and shipped with WinZip 9.0, is insecure in a number of natural scenarios. We exhibit several attacks and then propose ways of fixing the protocol. We believe that our proposed fixes to the Zip file format are relatively non-intrusive and that they will require only a moderate amount of reimplementation on the part of WinZip Computing, Inc. and the vendors of other WinZip-compatible applications.

We include this discussion of WinZip in this dissertation because the WinZip ap-

An earlier version of the material in this chapter appears in the Proceedings of the 11th ACM Conference on Computer and Communications Security [49], copyright the ACM.

plication has security vulnerabilities despite having a provably secure authenticated encryption scheme as its core (an Encrypt-then-MAC construction using AES in CTR mode for encryption and HMAC-SHA1 for message authentication). Our attacks do not violate the provable security of the Encrypt-then-MAC core, but rather exploit problems with the interface between this secure core and the rest of the WinZip system.

Our results serve to highlight both the limitations of and possible future directions for the provable security approach. First, our results show that the provable security a system's sub-component is, by itself, not sufficient to guarantee the security of the larger system. Rather, the designer of the larger system must take care when designing that system; for example, the designer of a larger system must ensure that the larger system establishes the proper preconditions for the correct use of the sub-component. As other examples, recall Bellare and Namprempre's [10] and Krawczyk's [52] attack against the generic Encrypt-and-MAC paradigm in Section 2.6.3 and our attack against a natural fix to the SSH authenticated encryption scheme in Section 3.4.

If we look closer at our results, however, there is a more positive conclusion. Namely, the makers of WinZip could have deflected many of our attacks if they had used the provable security approach to help them design the whole AE-2 system, rather than just incorporate the provably secure Encrypt-then-MAC sub-component into AE-2 in an *ad hoc* manner. Therefore, the results in this chapter highlight our belief that there is still much to gain by pushing the provable security approach further into real systems. This lesson is consistent with the other major thrusts of this dissertation, i.e., the work in Chapters 3 and 4 toward modeling and understanding realistic composition-based authenticated encryption schemes and the work in Chapter 5 on designing an authenticated encryption scheme around pragmatic constraints.

6.1 Overview

WinZip. We shall write "WinZip" when we mean "WinZip 9.0" or any other recent version of WinZip or a WinZip-compatible tool that uses the AE-2 authenticated en-

cryption scheme [83].[1] A WinZip archive can contain multiple files, and when that is the case, each file is encapsulated independently. For each file to archive, if the length of the file is above some threshold, WinZip first compresses the file using some standard compression method such as DEFLATE [31]. WinZip then invokes the AE-2 encryption method on the output of the previous stage. Specifically, it derives AES [28] and HMAC-SHA1 [52] keys from the user's passphrase and then encrypts the output of the compression stage with AES in counter (CTR) mode (AES-CTR) and authenticates the resulting ciphertext with HMAC-SHA1. As noted in Section 2.6.3, the underlying AES-CTR-then-HMAC-SHA1 core is a provably secure authenticated encryption scheme per results by Bellare and Namprempre [10] and Krawczyk [52] and standard assumptions on AES-CTR and HMAC-SHA1.

A collection of issues. All our attacks exercise different problems with the way that WinZip attempts to protect users' files. Furthermore, our attacks work in a variety of different settings, require a variety of different resources, and accomplish a variety of different goals, which means that different adversaries may prefer different attacks. Since no single "best" attack exists, since in order to eventually fix the protocol we first wish to understand the (orthogonal) security issues with the current design, and since we believe that each of the issues we uncover is informative, we discuss each of the main problems we found, and their corresponding attacks, in turn. We believe that our observations also serve to highlight the subtlety of cryptographic design since the WinZip AE-2 authenticated encryption method uses a provably-secure Encrypt-then-MAC core in a natural and seemingly secure way and since one of the attacks we discover was made possible because of the way that WinZip chose to fix a different problem with its earlier encryption method, AE-1.

The main issues we uncover include the following:

[1] According to the documentation packaged with WinZip 9.0, "Because the technical specification for WinZip's AES format extension is available on the WinZip web site, we anticipate that other Zip file utilities will add support for this format extension."

Information leakage. According to the WinZip documentation, there is a known problem with the WinZip encryption architecture in that the metadata of an encrypted file appears in the WinZip archive in cleartext. Contained in this metadata is the encrypted file's original filename, the file's last modification date and time, the length of the original plaintext file, and the length of the resulting ciphertext data, the latter also being the length of the compressed plaintext data plus some known constant. Although WinZip Computing, Inc. may have had reasons for leaving these fields unencrypted, the risks associated with leaving these fields unencrypted should not be discounted. For example, if the name of a compressed and encrypted file in the `PinkSlips.zip` archive is `PinkSlip-Bob.doc`, encrypting the files in the archive will not prevent Bob from learning that he may soon be dismissed. Additionally, a recent result from Kelsey [48] shows that an adversary knowing only the length of an uncompressed data stream and the length of the compression output will be able to learn information about the uncompressed data. For example, from the compression ratio an adversary might learn the language in which the original file was written [16]. Of course, the mere name, date, and size of the entire `.zip` archive may reveal information to an adversary, so the goal here should not be to prevent all information leakage, but to reduce the amount of information leakage whenever possible.

Interactions between compression and encryption. One of our chosen-ciphertext attacks exploits a novel interaction between WinZip's compression algorithm and the AE-2 Encrypt-then-MAC core. In particular, although the underlying AES-CTR-then-HMAC-SHA1 core of AE-2 provably protects both the privacy and the integrity of encapsulated data, an attacker can exploit the fact that the metadata fields indicating the chosen compression method and the length of the original file are *not* authenticated by HMAC-SHA1 as part of AE-2.

An example situation in which an adversary could exploit this flaw is the following: two parties, Alice and Bob, wish to use WinZip to protect the privacy and integrity of some corporate data. To do this, they first agree upon a shared secret passphrase.

Suppose Alice uses WinZip to compress and encrypt some file named `F.dat`, using their agreed upon passphrase to key the encryption, and let `F.zip` denote the resulting archive. Now suppose Alice sends `F.zip` to Bob, perhaps using WinZip's Outlook email plugin or by putting it on some corporate file server or an anonymous ftp server. We argue that the type of security that Alice and Bob would expect in this situation is very similar to the authenticated encryption notions of PRIV-CCA-privacy (Section 2.4) and AUTHC-integrity (Section 2.6).

Unfortunately, an adversary, Mallory, could break the security of WinZip under this model. For example, assume that Mallory has the ability to change the contents of `F.zip`, replacing it with a modified version, `F-prime.zip`, that has a different value in the metadata field indicating the chosen compression method and an appropriately revised value for the plaintext file length. When Bob tries to decrypt and uncompress `F-prime.zip`, he will use the incorrect decompression method, and the contents of `F.dat` upon extraction will not be the original contents of `F.dat`, but will now look like completely unintelligible garbage G. Now suppose that Mallory can obtain G in some way. For example, suppose Bob sends the frustrated note "The file you sent was garbage!" to Alice. If Mallory intercepts that note, he might reply to Bob, while pretending to be Alice, "I think I've had this problem before; could you send the garbage that came out so that I can figure out what happened; it's just garbage, there's no reason not to include it in an email." Mallory, after obtaining G, can reconstruct the true contents of Alice's original `F.dat` file.

We believe that the above attack scenario is realistic. It is the same scenario that Katz and Schneier [46] and Jallad, Katz, and Schneier [42] used when attacking email encryption programs and PGP, so any attack against WinZip's Outlook email plugin under the same scenario is at least as damaging; one difference is that our attack is applicable to WinZip in its default setting, whereas the previous attacks against PGP require the user to choose a non-default setting or to encrypt already compressed data. Even when users do not use WinZip's Outlook plugin to send encrypted attachments, we believe that there are other natural scenarios in which an adversary could mount

our attack. For example, employees of at least one large corporation, Diebold Election Systems, transported important election-related files, compressed and encrypted into Zip archives, via an anonymous ftp site [43].[2] Given Jones' [43] discussion of Diebold's procedures, we would be surprised if an adversary able to modify `F.zip` could not also get access to the decrypted, garbage-looking output G. Lastly, even if security-conscious users might try to prevent an adversary from learning G, we believe that security products should remain secure even in the face of potential misuses by non-security conscious users, which further suggests that the attack we describe is significant and should be protected against.

On the names of files and their interpretations. There are a number of systems that associate software applications with filenames; for example, a Microsoft Windows machine will by default open `.doc` files with Microsoft Word and `.ppt` files with Microsoft Power Point. Unfortunately, WinZip's AE-2 authenticated encryption method does not authenticate an encrypted file's filename metadata field, meaning that Mallory could modify the names of the encrypted files in an archive without triggering any detection mechanism within the extraction utility. This is problematic since, on a system like Microsoft Windows, it is important for an extracted file to have the same extension as the original file. Otherwise, when Bob tries to open that file, he will accidentally use the wrong application, get an error message, and thereby possibly allow Mallory to mount an attack similar to the one described in the previous heading. The issue described here is orthogonal to the issue of leaving an encrypted file's filename unencrypted; specifically, the issue is not that the filename is stored in cleartext, but that the filename is not authenticated, though also encrypting the filename would not hurt.

We discuss other issues that can arise from allowing an adversary to modify the names of encrypted files. The main lesson with all of these issues is that a file encryption utility must not only protect the integrity of the *contents* of an encrypted file, but must also protect the integrity of all of the *metadata*, like the filename or filename extension,

[2] These events preceded WinZip's invention of AE-2; Diebold used the traditional Zip encryption method.

necessary for the surrounding system to correctly interpret that data.

Interactions with AE-1 and a protocol rollback attack. According to the WinZip AE-2 specification [83], the AE-2 authenticated encryption method fixes a security problem with an earlier AE-1 authenticated encryption method. Further, according to [83], software implementing the AE-2 authenticated encryption method must be able to decrypt files encrypted with AE-1. While AE-2 does protect against a specific attack against AE-1, there is a protocol rollback attack against WinZip that exploits the fact that an adversary can force WinZip to use the AE-1 decryption method on an AE-2-encrypted file. The attack also exploits the fact that in addition to using HMAC-SHA1, AE-1 also uses a 32-bit CRC of the unencrypted file.

The attack works in the same setting as the previous attacks. In this attack, Mallory intercepts `F.zip`, makes a guess of the contents of `F.dat`, and creates a replacement `F-prime.zip` based off his guess. If Bob can successfully decrypt `F-prime.zip`, i.e., if Bob doesn't complain to Alice that the file failed to decrypt because of a failed CRC check, then Mallory learns with high probability whether his guess was correct. To compare this attack with the previous attack, note that Mallory only needs to learn whether `F-prime.zip` decrypted successfully. On the other hand, Mallory only learns whether his guess was correct. Still, this may constitute a serious attack if Mallory knows that the contents of `F.dat` is from a small set of possible values, perhaps because of pre-existing knowledge of the message space or additional information gleaned from the compression ratio, and wants to know which value it is. In some situations Mallory may learn more than just whether his guess was correct; details in Section 6.6.

Archives with encrypted and unencrypted files. According to the WinZip AE-2 specification, archives can contain both encrypted and unencrypted files. While this may have some functionality and usability advantages, there is also a rather serious security disadvantage. In particular, when a user invokes WinZip 9.0's extraction utility on an archive containing both encrypted and unencrypted files, WinZip 9.0 will ask for a passphrase. It will then proceed to extract *all* of the files in the archive, without telling

the user which files were encrypted and which were not. The user will thus think that all the files in the archive were encrypted (and authenticated), but, in fact, an adversary could have complete control over the contents of all but one of the files in the archive (one file must remain encrypted under the user's passphrase in order to force WinZip 9.0 to prompt the user for the passphrase). In Section 6.7 we provide evidence that suggests that although WinZip Computing, Inc. was unaware of the attack we found when they designed AE-2, other Zip manufacturers may have been aware of it, or at least knew that there were risks associated with allowing both encrypted and unencrypted files in Zip archives.

Key collisions and repeated keystream. When encrypting a file, WinZip first takes the user's passphrase and derives cryptographic keys for AES and for HMAC-SHA1. The key derivation process is randomized; one of the reasons for this randomization is so that two different files encrypted with the same passphrase will use different AES and HMAC-SHA1 keys. Unfortunately, because not enough randomness is used in the key derivation process, we expect AES key collisions after encrypting only 2^{32} files when using AES with 128-bit keys. Furthermore, the AE-2 specification says that the initial CTR mode counter is always zero.[3] Combining these two observations, we can expect CTR mode keystream reuse after encrypting only around 2^{32} files, which is much less than the 2^{64} files we would expect if we chose a different random key for each file. Additionally, assuming that the encrypted files are all of realistic size, then this is also less than the number of files we would expect if we used AES in CTR mode with just a single key but a randomly selected initial counter for each file.

Because WinZip encrypts each file in an archive independently, all 2^{32} files need not be put into separate archives; we expect keystream reuse even if all 2^{32} files are distributed amongst only a small set of WinZip archives. The problems with keystream

[3] Previously we said that the underlying Encrypt-then-MAC core of AE-2 is a provably PRIV-CCA- and AUTHC-secure authenticated encryption scheme per Bellare and Namprempre [10] and Krawczyk [52]. Because the initial CTR mode counter is always zero, we were assuming that each key is used to encrypt at most one message, which is typically the case with WinZip assuming that less than 2^{32} files are encrypted per passphrase.

reuse are well known: once Alice reuses keystream, Mallory will be able to learn information about the compressed and encrypted plaintext. In a worst-case scenario, if Mallory knew the entire content of the larger, after compression, of two files encrypted with the same keystream, then Mallory would immediately know the entire contents of the other file.

Other ways of attacking WinZip. There are other ways in which an adversary might attack WinZip or any other compression utility. For example, as noted in the WinZip documentation, an adversary might try to capture a user's passphrase by installing a keyboard logger on the user's computer or might try to resurrect a plaintext file from memory. We also observe what we believe to be a new integrity attack against self-extracting password-protected executables: an adversary wanting to replace the data encapsulated by a password-protected self-extracting executable could write a new executable, with a similar user interface to the real self-extracting executable, that asks for but ignores the user-entered passphrase and simply creates a data file of the adversary's choice. However, attacks such as these are unrelated to the AE-2 encryption method, and since our focus is on the AE-2 encryption method and WinZip's use of cryptography, we do not consider these attacks further.

Secure alternatives. In response to the cryptographic issues and attacks we found, we discuss a number of approaches for fixing the WinZip encryption method while simultaneously minimizing the changes to the AE-2 specification.

Other Zip encryption methods. There are a number of other passphrase-based Zip encryption methods besides WinZip's new AE-2. The traditional Zip encryption mechanism [40] has similar functionality to AE-2, but it has significantly worse security: the traditional Zip stream cipher has been broken [21, 77] and the contents of traditionally-encrypted archives can be efficiently recovered from the Zip archives directly, i.e., there is no need to mount a chosen-ciphertext attack like the ones we describe above.

PKWARE also recently announced a new passphrase-based encryption mechanism called EFS [65]. The January 2004 version of the PKWARE's EFS specification [66], as well as the traditional Zip encryption mechanism, are all vulnerable to our attacks that exploit generic properties of the Zip file format, namely the attacks exploiting (1) the information leakage of an encrypted file's metadata, (2) the fact that an encrypted file's filename is not authenticated, and (3) the fact that an archive can contain both encrypted and unencrypted files. Although the global applicability of issue (1) is by now folklore knowledge, and we have evidence to believe that some people, although unfortunately not WinZip Computing, Inc., may have known about some aspects of issue (3), we have seen no previous discussions of issue (2). The lack of previous discussions and awareness of these latter and other issues is likely because, until the creation of applications like Zip Outlook plugins, and until the publication of works like Katz and Schneier [46], the risks of chosen-ciphertext attacks were under-estimated.

The EFS specification [65], dated April 26, 2004 and appearing after the original release of the material in this chapter (IACR ePrint Report 2004/078), adds a new "filename encryption" feature that will encrypt the filename and other metadata fields of encrypted files. Although EFS's approach for addressing issue (1) is different than ours, and is an option that users or administrators may fail to turn on (it was not the default in the version we tested), we are pleased to find that our suggestions for fixing (1) are less intrusive to the Zip file format than PKWARE's (when "filename encryption" is turned on under PKWARE's new specification [65], PKWARE-encrypted archives are not parsable under the traditional Zip specification [40]). Unfortunately, PKWARE's new "filename encryption" feature alone cannot always fully protect against variants of our problems with issues (2) and (3), largely because encryption alone does not imply authentication. PKWARE's specification [65] also includes the ability to encrypt and sign files using public key cryptography, assuming the presence of the requisite additional infrastructure, though it is worth noting that the "certificate processing method for ZIP file encryption remains under development ... and is subject to change without notice [65]."

Although a full treatment of PKWARE's new EFS passphrase-based encryption mechanism, as well as PKWARE's use of public key cryptography, is outside the scope of this chapter, we make a few observations here. The passphrase-based encryption mechanism does not include a message authentication code at all, and thus does not appear to have been designed to protect the privacy or integrity of files under chosen-ciphertext attacks. This is problematic since, although digital signatures can be used to protect the authenticity of the encapsulated data, it is still important to protect the authenticity of files encrypted with passphrases when the necessary infrastructure for digital signatures is not available, or when a user does not want to be bound to the contents of a file with a digital signature. The specification is also incomplete, making it not only difficult to implement the system from the specification alone, but to fully analyze the system for potential security problems without making conjectures about how the system is actually supposed to behave; e.g., if the user or developer chooses RC4 for encryption, how exactly is RC4 supposed to be used and are results like Mironov's [61] taken into consideration? Where the specification is unambiguous, the specification still leaves decisions, such as the choice of the underlying cipher (e.g., 40-bit RC2, 64-bit RC4, 3DES, AES) and the length of the randomness RD when deriving encryption keys, up to the choice of implementors. This is a concern since even if PKWARE makes safe choices with respect to these decisions, there is nothing in the specification to prevent third-party developers from making unsafe choices.

Additional related works. Biham [20] introduced the notion of key-collision attacks in the context of DES, noting that we expect one key collision after encrypting about 2^{28} messages using randomly selected 56-bit DES keys; our keystream reuse attack in Section 6.8 is related to Biham's key-collision attack except that it is more efficient than a normal key collision attack because of the way that WinZip derives AES keys from passphrases. Wagner and Schneier discuss protocol rollback attacks in [80].

6.2 The WinZip Compression and Encryption Method

WinZip's compression architecture follows the Info-ZIP specification [40]. The AES-based AE-2 extension is described on WinZip's website [83]. The difference between the AE-2 authenticated encryption method and the AE-1 authenticated encryption method is slight and is mentioned at the end of this section.

Basic structure. We present here the basic Zip file format and the AE-2 extensions, omitting details that are not relevant to our attacks and to our security improvements.

A Zip archive can contain multiple files. When archiving a set of files, WinZip creates two *records* for each file, a *main file record* and a *central directory record*. The resulting Zip archive contains all of the main file records concatenated together followed by all of the central directory records. Following the central directory records is an *end of archive record*, which is not relevant to our attacks and suggested improvements. The *main file record* contains metadata about the file, like the filename, as well as the file's contents, the latter typically being compressed and, in the case of AE-2, encrypted. The contents of each file is compressed and encrypted independently. The *central directory record* mirrors the metadata stored in the main file record and also contains information about the location of the file's corresponding main file record in the Zip archive. One of the reasons for the existence of the central directory record is for usability when working with multi-volume floppy or CD archives. For example, when extracting a file from a multi-volume CD archive, the user can insert the last CD, WinZip can read the central directory information, and then WinZip can prompt the user to insert the CD containing the main file record.

When referring to the fields of a Zip archive, byte strings will be written like $\texttt{504b0304}_{\text{bs}}$, meaning that the first byte is $\texttt{50}_{\text{bs}} = 80$, the second byte is $\texttt{4b}_{\text{bs}} = 75$, and so on. Integers, such as lengths, that are stored in multi-byte fields are encoded in little endian format.

Main file record. According to the Info-ZIP specification [40], and barring certain extensions that do not affect our attacks, all main file records have the following structure (the fields important to our work are highlighted): main file record indicator (4 bytes, always $\texttt{504b0304}_{\text{bs}}$), version needed to extract (2 bytes), general purpose bit flag (2 bytes), *compression method* (2 bytes), *last modification time* (2 bytes), *last modification date* (2 bytes), *32-bit CRC* (4 bytes), compressed size (4 bytes), *uncompressed size* (4 bytes), filename length (2 bytes), extra field length (2 bytes), *filename* (variable size), and *extra field* (variable size). Following the above fields, but still part of the main file record, is the *file data* field.

Central directory record. The central directory record for a file consists of the following fields (important fields highlighted): central directory record indicator (4 bytes, always $\texttt{504b0102}_{\text{bs}}$), version made by (2 bytes), version needed to extract (2 bytes), general purpose bit flag (2 bytes), *compression method* (2 bytes), *last modification time* (2 bytes), *last modification date* (2 bytes), *32-bit CRC* (4 bytes), compressed size (4 bytes), *uncompressed size* (4 bytes), filename length (2 bytes), extra field length (2 bytes), file comment length (2 bytes), disk number start (2 bytes), internal file attributes (2 bytes), external file attributes (4 bytes), relative offset of local header (4 bytes), *filename* (variable size), *extra field* (variable size), and file comment (variable size).

AE-2 settings and the AE-2 extra data field. The following is applicable to both the main file record and the central directory record. When the AE-2 WinZip encryption algorithm is turned on, the four bytes reserved for the 32-bit CRC are set to zero, bit 0 of the general purpose flag is set to 1, and the two bytes reserved for the compression method are set to $\texttt{6300}_{\text{bs}}$. The extra data field will consist of the following 11 bytes (again, important fields highlighted): extra field header id (2 bytes, always $\texttt{0199}_{\text{bs}}$), data size (2 bytes, $\texttt{0700}_{\text{bs}}$ for AE-2 since there are seven remaining bytes in the 11-byte extra data field), *version number* (2 bytes, always $\texttt{0200}_{\text{bs}}$ for AE-2), 2-character vendor ID (2 bytes, always $\texttt{4145}_{\text{bs}}$ for AE-2), value indicating AES encryption strength (1 byte), and *the actual compression method used to compress the file* (2 bytes). The encryption

strength field will be $\texttt{01}_{\text{bs}}$ (resp., $\texttt{02}_{\text{bs}}$ or $\texttt{03}_{\text{bs}}$) if the file is encrypted with AES using a 128-bit (resp., 192-bit or 256-bit) key. Example values for the actual compression method are $\texttt{0800}_{\text{bs}}$ if the file is DEFLATEd [31] and $\texttt{0000}_{\text{bs}}$ if no compression is used.

File data field. When a file is AE-2-encrypted, the file data field of the main file record contains the following information: *salt* (variable length), *password verification value* (2 bytes), *encrypted file data* (variable length), and the *authentication code* (10 bytes). The salt is 8 bytes (resp., 12 bytes or 16 bytes) long if the AES key is 128 bits (resp., 192 bits or 256 bits) long.

The encrypted file data and authentication code. Before applying the AE-2 authenticated encryption method, the contents of the plaintext file is compressed according to the "actual compression method used to compress the file" field of the AE-2 extra data field described above. Then an AES encryption key, an HMAC-SHA1 key, and a password verification value are derived from the user's passphrase and a salt using the PBKDF2-HMAC-SHA1 algorithm [45]. The length of the salt depends on the chosen length of the AES key and is described above. The specification [83] states that the salt should not repeat, and since this must be true across different invocations of the compression tool, suggests making the salt a random value.

The derived AES key is used to encrypt the compressed data using AES in CTR mode with the initial counter set to zero. The compressed plaintext data is not padded before encryption. After encryption, the encrypted data is MACed using HMAC-SHA1 and the derived MAC key, and 80 bits of the HMAC-SHA1 output are used as the authentication code.

Differences between AE-1 and AE-2. The only differences between the AE-2 method and the earlier AE-1 method is that in AE-1 the version number in the main file record's and central directory record's extra data fields are $\texttt{0100}_{\text{bs}}$ and the 32-bit CRC fields are not all zero but actually contains the CRC of the original unencrypted data, which the WinZip specification [83] states must be checked upon extraction. The

motivation for zeroing out the CRC field in AE-2 is because the CRC of the plaintext will leak information about the plaintext.

6.3 Information Leakage

The metadata fields of encrypted files leak important and potentially security-critical information in several ways. The names of the encrypted files are stored in cleartext, which can obviously be a concern. The files' last modification dates and times are also stored unencrypted, which can be used to infer some relationship between the contents of different encrypted files or some event in the past. Additionally, the length of plaintext files are stored in the files' metadata fields unencrypted. This is a concern since, based on Kelsey's recent results about compression as a side-channel [48], an adversary can learn information about the plaintext simply given the lengths of both the original and the compressed data. As Kelsey notes, information leakage via the compression ratio of files becomes particularly effective if Mallory has pre-existing partial knowledge of the plaintext or if Mallory can see the compression ratio of multiple related files, e.g., different versions of the same file over time. The WinZip documentation notes that these pieces of information are included unencrypted in the file's metadata, but the risks associated with leaving these fields unencrypted is not considered. Furthermore, many users may fail to read the documentation, and thus may not realize that these information leakage side-channels exist in the first place.

It is a well known fact that the classic Zip encryption method [40] also leaks the information that we mention above, plus the 32-bit CRC of an encrypted file's original plaintext. It is interesting to ask why WinZip Computing, Inc. did not fix this problem in their new AE-2 specification. The most likely conjecture is that WinZip Computing, Inc. chose not to do so either because of engineering or design complexities, or because of functionality issues (e.g., they wanted to allow users to be get a directory listing of the contents in their encrypted archives without having to enter a passphrase). To address the former reason, we discuss technical approaches for addressing the information leakage

concerns in Section 6.10.

6.4 Exploiting the Interaction Between Compression and Encryption

Recall the setup described in Section 6.1, where Alice encrypts `F.dat` and sends the resulting Zip archive, `F.zip`, to Bob, but where Mallory prevents the delivery of `F.zip` and instead gives Bob a file, `F-prime.zip`, that is related to `F.zip` but that is slightly different. The critical observation for our attack is that despite the fact that the underlying encryption core is a provably secure Encrypt-then-MAC authenticated encryption scheme, the compression method and original file length fields in an encrypted file's main file and central directory records are *not* authenticated, which means that an adversary can change these fields without voiding the HMAC-SHA1 authentication tag attached to the file. Consequently, assuming that the new uncompressed file length field is correct or that the extraction tool does not check that field, when Bob attempts to decrypt and decompress the modified file `F-prime.zip`, the MAC verification will succeed and WinZip will not report any errors. But because the adversary changed the compression method, the file will be decompressed using the wrong algorithm and the resulting contents G of the extracted file will look like garbage. This issue immediately violates the type of security goal captured by the AUTHC definition in Section 2.6. If Mallory can learn G, which we argue in Section 6.1 is reasonable in some cases, Mallory can recover the original contents of Alice's file `F.dat`. This latter step, in addition to being of concern in practice, violates the type of security goal captured by the PRIV-CCA definition in Section 2.4.

Implementing the attack. When mounting the attack, Mallory would likely change the compression method indicators in the main file and central directory records from $\texttt{0800}_{\text{bs}}$, which appears to be WinZip's default and which corresponds the DEFLATE algorithm [31], to $\texttt{0000}_{\text{bs}}$, which corresponds to no compression. This is very easy to

do and very efficient and can be done in a linear pass through the file, as can updating the original file length field. We implemented this attack against WinZip 9.0. To create `F-prime.zip` from `F.zip`, rather than parse `F.zip` and switch the compression type from $\texttt{0800}_{\text{bs}}$ to $\texttt{0000}_{\text{bs}}$, we found that the Unix `tcsh` command line

```
cat F.zip |\
  sed 's/\(\x02\x00\x41\x45\x01\)\x08\x00/\1\x00\x00/g'\
  > F-prime.zip
```

was sufficient in all of the cases that we tried, showing that the attack is indeed very easy to mount.[4] We would only expect the above command line to not work as desired if the 7-byte string $\texttt{0200414501080}_{\text{bs}}$ appears in `F.tar` in a place not corresponding to the extra data field of a file's main file or central directory records. Since the WinZip 9.0 extraction tool did not seem to verify the length of the extracted file, we did not need to modify the original file length fields of the file's main file and central directory records.

Subtlety of cryptographic design. Recall that in AE-1 the CRC field of an encrypted file's header contains the CRC of the original plaintext file but that the field is all zero in AE-2. When trying to mount the above attack against AE-1, since the extraction utility will also verify the CRC of the plaintext, which will typically fail because the plaintext is now different, the resulting garbage-looking data G will not be saved and the attack will not immediately go through. While it is true that if Bob is crafty he may be able to view `F.dat` (the file with contents G) among the temporary files created by WinZip during the extraction process and before the CRC failure is noted, send G to Alice, and thereby leak G to Mallory, it might be unrealistic for Mallory to assume that Bob will find `F.dat` among WinZip's temporary files, at least not without more active intervention by Mallory. This discussion highlights the subtlety of cryptographic design since the vulnerability presented in this section was accidentally introduced when the authors of the specification tried to fix a different problem with AE-1.

[4] Different versions of `sed` appear to handle binary streams differently. The attack worked on default RedHat 9.0 systems with `sed` version 4.0.3.

6.5 Exploiting the Association of Applications to Filenames

To complement the attack in Section 6.4, we note that on many systems, including Microsoft Windows machines, software applications are automatically attached to files based on the files' filename extensions; e.g., Microsoft Windows will by default open `.doc` files with Microsoft Word. Since the filename fields of an encrypted file's main file and central directory records are unauthenticated, an adversary could modify those field without voiding the MAC included at the end of the encrypted file's main file record. Once Mallory does this, he can mount a variant of the attack in Section 6.4 since applications will usually report an error when trying to open a file of the wrong extension. Fortunately, some applications give descriptive error messages and Bob may realize that the file has the wrong filename extension (e.g., Microsoft Excel gives the error "`File.xls`: file format is not valid" when opening a document created with Microsoft Word), but this is largely serendipitous and should not be relied upon for security. This discussion suggests that a file encryption utility must not only protect the integrity of the encapsulated data itself, but also the metadata, like the filename extension, necessary for the surrounding system to correctly interpret that data.

We also observe that an adversary could benefit from changing the names of the encrypted files in an archive while still maintaining the files' original extensions. For example, if Alice's salary is currently higher than Mallory's, Mallory could swap the names of the files `Alice-Salary.dat` and `Mallory-Salary.dat` in an encrypted archive `Salaries.zip` without triggering any detection mechanism within the extraction utility.

6.6 Exploiting the Interaction Between AE-1 and AE-2

The motivation for the change from AE-1 to AE-2 is that in AE-1 the CRC of the plaintext file is included unencrypted in an AE-1-encrypted WinZip archive, and that

will leak information about the encrypted files' contents. While the CRC is no longer included in the output of the AE-2 authenticated encryption method, one can exploit an interaction between AE-1 and AE-2 in the following PRIV-CCA-style attack that reveals information about an AE-2-encrypted file's CRC to an adversary. Our attack makes use of the fact that, according to the AE-2 specification [83], Zip tools that understand AE-2 must be able to decrypt files encrypted with AE-1 and must verify the CRC upon extraction.

Details. Recall the PRIV-CCA-based setting used in Section 6.4 and Section 6.5. Assume Alice sends the encrypted file `F.zip` to Bob, but assume that Mallory can modify the file in transit and can learn whether Bob can successfully extract the file he receives using the passphrase he shares with Alice. Now suppose that Mallory has a guess for what the original contents of `F` are, but is not completely sure and wants to verify his guess H. He can do this as follows: compute the 32-bit CRC of H and then modify `F.zip` such that the version number in the main file and central directory records' extra data fields are $\texttt{0100}_{\text{bs}}$ and the CRC fields in the file's main file and central directory records has the CRC of H. Let `F-prime.zip` denote the Mallory-doctored file. If Mallory's guess is correct, then Bob will be able to extract `F` from `F-prime.zip` without any error. Otherwise, Bob will with high probability see an error dialog box which, when using WinZip 9.0, says "Data error encountered in file `C:\F`[.] Possibly recoverable, contact `help@winzip.com` and mention error code 56." By observing Bob's reaction, Mallory will with high probability learn whether his guess was correct.

If we look more closely at how WinZip behaves when it attempts to extract a modified file with an incorrect CRC guess, it appears that the file is first extracted, the CRC is checked, the user is told that the CRC check failed, and then the extracted file is deleted. This means that if Bob is crafty he will be able to access the unencrypted file between when it is extracted and when it is automatically deleted after the CRC check fails. Even if Bob does this, which we expect to be unlikely, he may not be confident in the correct extraction of the file and, if so, will likely convey this lack of confidence

to Alice. Other implementations of the AE-2 specification may delete the extracted file before informing the user that the CRC check failed.

Extension. Although not necessarily the case with all Zip tools but in the case of WinZip, after dismissing the initial error dialog box Bob will have the option of viewing a more detailed error log. If Bob chooses to see this error log, he will see a line like the following:

```
bad CRC 1845405d  (should be 1945405d)
```

If Bob decides to copy and paste this detailed error message in an email to Alice or `help@winzip.com`, and if Mallory sees this email, then Mallory will learn the CRC of the plaintext file, and thereby learn additional information about the plaintext.

6.7 Attacking Zip Encryption at the File Level

When a Zip archive contains multiple files, each of the files in the archive is encapsulated independently, which means that some files in an archive may only be compressed and some may be both compressed and encrypted. This fact makes the WinZip AE-2 authenticated encryption method vulnerable to a number of attacks. Consider the following: Mallory knows that the encrypted archive `Salaries.zip` contains the files `Alice.dat`, `Bob.dat` and `Mallory.dat`, all encrypted using AE-2 under the CFO's secret passphrase. Now, because of the properties described above, an adversary could remove the encrypted `Mallory.dat` file from the `Salaries.zip` archive and replace it with a *new, unencrypted* file, also named `Mallory.dat`, but with the contents of Mallory's choice. When the CFO tries to extract the files in the archive using the WinZip 9.0 application, she will be prompted for her passphrase since the files `Alice.dat` and `Bob.dat` are still encrypted. WinZip will then extract the files `Alice.dat`, `Bob.dat`, *and* `Mallory.dat`. Since the CFO had to enter her passphrase, she will likely believe that the extracted `Mallory.dat` file is the same one that she encrypted, and thus contains Mallory's real salary, when in fact the contents of

`Mallory.dat` are completely under Mallory's control. Similarly, if Alice creates an archive containing both encrypted and unencrypted files and sends that archive `F.zip` to Bob, Mallory will be able to easily modify the contents of the unencrypted files in the archive. But, like in the previous attack, since Bob has to enter a passphrase to extract the contents of the archive, and because no warning is given about some files being unencrypted, Bob will believe that all the files were encrypted by Alice and that they contain Alice's original content.

WinZip Computing, Inc. does not appear to have been aware of the above attacks when they specified AE-2 [83] and when they implemented WinZip 9.0, as supported both by the fact that WinZip 9.0 does not generate a warning when extracting an archive containing both encrypted and unencrypted files, and by quotes taken from the AE-2 specification [83], which only mention usability reasons for encrypting all the files in an archive and which do not suggest that vendors issue warnings when encountering unencrypted files in an archive with encrypted files. E.g., the specification states: "The presence of both encrypted and unencrypted files in a Zip [archive] may trigger user warnings in some Zip file utilities, so the user experience may be improved if all files (including zero-length files) are encrypted. Again, however, this is only a recommendation." This quote does suggest that other Zip vendors may have known of the attack we describe above, or at least knew to be wary of archives containing both encrypted and unencrypted files.

Because files in a Zip archive are encrypted on a per-file basis, an adversary could also delete files from an archive. An adversary could also create a composite Zip archive with encrypted files taken from multiple different archives, but we view these properties as less interesting than the first attacks in this section. Related to the first attacks in this section, in Section 6.5 we observed that an adversary could swap the filenames of different encrypted files, and that he could also use this fact to modify the contents of Alice's encrypted files; the attacks in Section 6.5 exploit a different security problem, that for encrypted files the filenames are not authenticated.

6.8 Keystream Reuse

When AE-2 is used with a 128-bit AES key, one can expect CTR mode keystream reuse after encrypting approximately 2^{32} files, which is much less than one would expect given that AES has 128-bit blocks. (When using 192-bit AES keys with AE-2, we expect keystream reuse after encrypting 2^{48} files; when using 256-bit AES keys, we expect collisions after encrypting 2^{64} files). The security problems with reusing keystream are well-known, and therefore we can expect the AE-2 authenticated encryption method with 128-bit AES keys to start leaking additional information about the compressed and encrypted plaintext after 2^{32} files are encrypted with the same passphrase.

This problem arises for two reasons. First, the salt used when deriving the AES and HMAC-SHA1 keys from the passphrase is only 64 bits (resp., 96 bits and 128 bits) long when the desired AES key length is 128 bits (resp., 192 bits and 256 bits). Second, AES-CTR is specified to always use zero as the initial block counter. The former means that, with 128-bit keys, after encrypting 2^{32} files we expect there to be one AES key that we used twice. The latter means that when we use the same AES key twice, we will use the same keystream both times.

6.9 Dictionary Attacks

One of the reasons for using PBKDF2 [45] and a salt when deriving AES and HMAC-SHA1 keys from passphrases is to impede dictionary attacks. Specifically, an exhaustive search through the most common passphrases will be slow because of the computational requirements for PBDKF2, and a dictionary of HMAC-SHA1 keys, corresponding to the most common passphrases and all possible salt values, will be extremely large because of the number of possible salt values.

But since a different salt is used to encrypt each file, an adversary may not need to use *all* possible salt values when populating an HMAC-SHA1 key dictionary. In particular, Mallory would only need to populate the dictionary using enough different salt values to ensure, with high probability, that one of the salt values that a user uses when

encrypting her files will collide with one of the salt values that Mallory used when creating his dictionary. For example, if the salt is 8 bytes long and if each user is expect to encrypt on the order of 2^{32} files, then Mallory would only need to use 2^{32} different salt values when creating his HMAC-SHA1 dictionary. The dictionary can be indexed off of the salt *and* the two-byte password verification value; the password verification value thus further reduces the amount of HMAC-SHA1 keys the attacker has to try in the dictionary attack. Once Mallory finds an HMAC-SHA1 key such that the MAC of the encrypted file verifies, he will with high probability learn the user's corresponding passphrase, and thereafter be able to decrypt all of the files encrypted under that passphrase. While this is a time-memory trade-off in terms of not having to compute PBKDF2 for every passphrase guess, the memory and precomputation requirements are still quite enormous.

6.10 Fixes

In this section we consider fixes to the problems we discussed in Section 6.3 through Section 6.9, starting with Sections 6.4–6.9 and returning to Section 6.3 at the end. We also discuss our preferred instantiations of these suggestions.

Authenticate all. To address the problems raised in Section 6.4, one approach might be to MAC the original uncompressed plaintext instead of the ciphertext and then encrypt the resulting tag in a MAC-then-Encrypt-style construction. We recall from Section 2.6.3 and Chapter 4 that, while MAC-then-Encrypt is not generically secure, it is possible to base secure authenticated encryption schemes on the MAC-then-Encrypt paradigm. Alternatively, we could build on WinZip's current provably secure Encrypt-then-MAC core. If we continue to use the existing Encrypt-then-MAC core, we still note the following general design principle for cryptographic encapsulation methods: a cryptographic encapsulation algorithm should authenticate *all* of the information that an extractor/decapsulator will use when reconstructing the original data, excluding the

authentication tag itself and assuming that the extractor already has a copy of the shared authentication key. In the case of WinZip, since the compression type field of an encrypted file's header will be accessed when extracting an encrypted file, this means that the compression type value should be MACed along with the AES-CTR-generated ciphertext. We can naturally extend this general principle to mandate the authentication of all data necessary to ensure the correct *interpretation* of the data once the data has been correctly reconstructed, which means that the filename, date, and any other important metadata fields in an encrypted file's header must also be authenticated, which addresses the concerns raised in Section 6.5. If WinZip Computing, Inc. does not mind deviating further from their current AES-CTR-then-HMAC-SHA1 construction, then the new encryption core can be any provably-secure AEAD scheme as long as the important metadata fields are authenticated.

Addressing protocol rollback attacks. To prevent protocol rollback attacks like the one described in Section 6.6, it might be tempting to apply the above principle and create a new scheme that MACs the encryption method version number field in the extra data field of an encrypted file's header. Unfortunately, this may not necessarily work since here we are concerned about attacks that exploit the interaction between different encapsulation/decapsulation schemes, and, in particular, interactions with schemes, AE-1 and AE-2, that have already been specified and that do not currently authenticate that field. To see why this is a problem, note that an adversary could move the extra data MACed using the new method into the ciphertext portion of an AE-2-format archive and thereby mount a protocol rollback attack.

While one might try MACing information not directly available to an adversary, such as the encipherment of some nonce, we view such an approach as inelegant. Rather, we suggest diversifying the AES and HMAC-SHA1 key derivation process in such a way that the AES and HMAC-SHA1 keys derived from some passphrase and salt using the new encryption method will be different from the keys derived from the same passphrase and salt when using the AE-1 and AE-2 encryption methods. This could

involve prepending the encryption method version number, vendor ID, and encryption strength field to the salt before running the key derivation procedure. If it were not the case that the length of the salt for AE-1 and AE-2 were fixed, but if the length of the salt was variable and if the length of the salt is encoded in a metadata field of an encrypted file, then even our solution here would not be a sufficient since an adversary could simply add the method version number, vendor ID, and encryption strength field into the (now larger) salt in an AE-2-formatted archive. For similar reasons, there is still the potential of interaction with other (non-WinZip) applications that uses PBKDF2-HMAC-SHA1, but it seems impossible for WinZip to complete avoid such interactions with applications that are not under their control.

Addressing the concerns in Section 6.7. There are several possible solutions for the problems that we raised in Section 6.7. The obvious approach of authenticating an entire archive would likely break some of WinZip Computing, Inc.'s functionality design criteria, namely the desire to (efficiently) handle updates to large archives, and in particular archives spanning multiple CD volumes. Another approach might be to authenticate the entire central directory (the concatenation of all the central directory records), since the central directory will always be stored at the end of the archive, and in particular on the last CD in a multi-volume archive. Toward this end, we note that the Zip specification already has the ability to sign the central directory using public key cryptography, so adding the ability to authenticate the central directory using a MAC is certainly reasonable. However, we point out that this solution has a number of issues that one must be careful of. For example, the extractor must check the consistency between the metadata in a file's main file record and a file's central directory record. If we are concerned about adversaries deleting files from an archive, then the absence of files must also be checked (this may follow as a corollary of checking the consistency of the individual files if the consistency check includes main file record offsets, which are stored in the central directory record). But of most concern is the fact that authenticating the central directory alone will *not* prevent an attacker from modifying unencrypted files in an

archive. Rather, those unencrypted files must be cryptographically bound to the central directory in some way, perhaps by including a MAC of an unencrypted files in its central directory record. Another potential problem with this solution is that *if* authenticating the central directory is an option, then one must be careful to ensure that an adversary cannot simply take a Zip archive, turn that option off, and remove the MAC of the central directory. One possible way of handling this might be to use different AES and HMAC-SHA1 keys when the option is turned on and when the option is turned off. Alternatively, a reasonable solution might simply be to *require* applications implementing the AE-2 decryption algorithm to *always* report a warning when an archive contains both encrypted and unencrypted files.

Addressing keystream reuse and dictionary attacks. To address the issues raised in Section 6.8, we suggest two possible solutions. First, one could double the current salt length. Alternatively, instead of always using zero as the initial AES-CTR mode counter, one could use a random initial counter selected from the set of all possible 128-bit integers. The initial counter should be included in the resulting archive and should also be included in the string to be MACed. Furthermore, under this approach the same AES and HMAC-SHA1 keys can be used with all files protected by the same passphrase; i.e., the same randomly-selected salt could be used with all such files in an archive. The latter property is a performance gain since in the current design, where a different salt is used with each file, the passphrase-based key derivation step dominates the time when creating or extracting archives containing lots of small files. When adding new files to an existing archive, it is important to select new salts or to verify that the users knows the passphrase corresponding to the files encrypted with the existing salt values (otherwise an attacker could force a user to use a salt of the attacker's choice, which would make dictionary attacks more feasible).

Possible solutions to the issues raised in Section 6.9 include increasing the length of the salt or using the same salt when encrypting multiple files. Fortunately, these two recommendations align with our recommendations for the issues raised in Section 6.8.

Additionally, we suggest not storing the password verification values in a file's metadata since it can be used to quickly eliminate keys in a dictionary attack against a user's passphrase.

Minimizing information leakage. There are a number of different approaches for addressing the information leakage concerns raised in Section 6.3. The latest (April 26, 2004) specification from PKWARE [65], which is incompatible with WinZip's new encryption method, introduces an option for encrypting the metadata fields of an encrypted file; when the option is turned on (it is not on by default), PKWARE's SecureZIP product encrypts the entire central directory and removes most of the metadata information from a file's main file record, either by zeroing out the appropriate fields or replacing them with random data. Aside from the fact that the central directory is not MACed, our two main concerns with PKWARE's solution are that (1) we believe that protecting against information leakage from an encrypted file's header should not be an option and (2) archives created with the above option turned on are no longer parsable under the traditional Zip specification [40]. In contrast, our proposed fixes involve modifying the main file and central directory records such that privacy-critical metadata information is always hidden and the resulting Zip archives are still parsable under the traditional Zip specification [40].

We can achieve this goal in several ways. For example, using AES in CTR mode, it would be possible to encrypt specific metadata fields of a file's main file record and central directory record in-place. In the case of the central directory record, this approach would require us to copy the salt necessary to derive the encryption key from the file data field of the main file record into the extra data field of the central directory record. Unfortunately, this solution must still leak the length of a file's filename since, under this approach, we cannot encrypt any information necessary for parsing the file, and the length of a file's filename is necessary information.

The solution that we prefer is to not encrypt portions of a file's main file record and central directory records in-place, but to encrypt (and also authenticate) the main

file record and the central directory record completely. Our solution would then store the resulting ciphertext in the file data or extra data fields of a wrapper main file record or wrapper central directory record, respectively. Preceding the ciphertexts must be the information, like the salt, necessary to derive the file's cryptographic keys from the user's passphrase. The metadata fields of these wrapper records can be fixed, or random, as long as the "compression method field" in the main file record indicates that the record is just serving as a wrapper for an encrypted file. When extracting an archive, the extractor should see this specific compression method type, decrypt the wrapped data, and then treat the resulting plaintext as an unencrypted record to parse as normal.

In order to give an intuitive error message to users who try to decrypt a file encrypted under this method, we suggest making the filename field of the wrapper records something like `WinZipEncryptedFile`; one could even add more information, like a URL. Lastly, another attractive property of this solution is that, by also authenticating these records completely, this solution immediately implements our previous recommendations for addressing the concerns in Section 6.4 and Section 6.5.

A possible instantiation. Given the recommendations made in the above paragraphs, one possible instantiation might be the following, which is based on AE-2 but which we call BE since it is different enough to warrant a new name. For each file to archive, compress the file and create main file and central directory records as if encryption was not used. Then select a random value the same length as the salt in AE-2, concatenate information about the encryption scheme (BE algorithm identifier, version number, and AES-key-length value) with the random value, and call the resulting value the salt for BE. Derive AES and HMAC-SHA1 keys from the user's passphrase and the salt using PBKDF2-HMAC-SHA1. Then use that AES key to CTR mode encrypt all of the main file and central directory records, using a randomly selected initial counter (IV) for each record (the main file and the central directory records for a single file should have different random IVs). Then MAC the IVs concatenated with each of the ciphertexts using HMAC-SHA1. Then concatenate the BE algorithm identifier, version number,

AES-key-length, the random value in the salt, the CTR mode IV, the ciphertext, and the MAC for each record. No password verification value is stored in these resulting strings. For the resulting string consisting of the encryption of the main file record, load it into the data portion of a wrapper main file record that has bit 0 of the general purpose flag set to 1 (meaning that the file is encrypted) and that has a "compression method" field indicating that the file is encrypted under our new encryption method; the other fields can be anything that does not leak information about the wrapped file. For the resulting string consisting of the central directory record, load it into the extra data portion of a wrapper central directory record that has the same general purpose flag and compression method as for the wrapper main file record.

When extracting an archive, the user must be warned whenever encountering an unencrypted file in an archive with encrypted files. The MAC must also be checked during decryption. Although all the data necessary to reconstruct a file is stored in the file's wrapped main file record, we still maintain the central directory record since it is part of the classic Zip file format [40] and since it will be used by some parties to quickly find specific files in an archive. If there are inconsistencies between a file's pair of records, an error should occur.

Although the same random value in the salt can be used for multiple files when encrypting them all at once, a new random value should be chosen if the user decides to update a file or add a new file to an archive. Alternatively, when updating a file or adding a new file to an archive, if one wants to use the same random value in the salt as before, they must check that the user's passphrase combined with the existing salts successfully decrypts currently-encrypted files. If either of these solutions were not in place, then an adversary could replace the random values in the salts in an archive with any value of his choice, and create a dictionary of AES and HMAC-SHA1 keys corresponding to the single chosen salt value. Additionally, when changing the contents of the file, and to avoid keystream reuse, a new random initial counter for CTR mode must be selected.

The security of this construction follows from the earlier discussions in this section and the provable security of AES-CTR-then-HMAC-SHA1; unlike with AE-2, we can

employ Bellare and Namprempre's [10] and Krawczyk's [52] positive results on the generic Encrypt-then-MAC paradigm when discussing BE since we are now encrypting *all* the data of interest, rather than just a portion of it. The risks associated to AES key collision attacks are minimized by the use of a random IV in AES-CTR (specifically, AES key collisions no longer immediately imply keystream reuse). BE can still leak information from the compression ratio of a file if the adversary knows the original length of the file (the original length is now no longer visible directly from the archive itself); this is acceptable because we are unaware of any solution to the information-leakage-through-compression problem without adding additional padding and thereby reducing some of the space savings generally associated with compression. Our new method is more efficient than AE-2 when adding multiple files to an archive in batch, or extracting multiple archives from a file in batch; this is because PBKDF2 is intentionally slow by design and, unlike AE-2, BE only invokes PBKDF2 once for all files added to an archive at the same time.

Additional Information

An earlier version of the material in this chapter appears in the Proceedings of the 11th ACM Conference on Computer and Communications Security [49], copyright the ACM. I was a primary researcher and single-author on this paper. The full citation for this work is:

> Tadayoshi Kohno. Attacking and repairing the WinZip encryption scheme. In Birgit Pfitzmann, editor, *Proceedings of the 11th ACM Conference on Computer and Communications Security*, pages 72–81. ACM Press, October 2004.

Bibliography

[1] J. H. An and M. Bellare. Does encryption with redundancy provide authenticity? In B. Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 512–528. Springer-Verlag, Berlin Germany, 2001.

[2] M. Bellare. New proofs for NMAC and HMAC: Security without collision-resistance. In C. Dwork, editor, *Advances in Cryptology – CRYPTO 2006*, Lecture Notes in Computer Science. Springer-Verlag, Berlin Germany, 2006.

[3] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In N. Koblitz, editor, *Advances in Cryptology – CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, Berlin Germany, Aug. 1996.

[4] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, pages 394–403. IEEE Computer Society Press, 1997.

[5] M. Bellare, R. Guérin, and P. Rogaway. XOR MACs: New methods for message authentication using finite pseudorandom functions. In D. Coppersmith, editor, *Advances in Cryptology – CRYPTO '95*, volume 963 of *Lecture Notes in Computer Science*, pages 15–28. Springer-Verlag, Berlin Germany, Aug. 1995.

[6] M. Bellare, J. Kilian, and P. Rogaway. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences*, 61(3):362–399, 2000.

[7] M. Bellare and T. Kohno. A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications. In E. Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 491–506. Springer-Verlag, Berlin Germany, May 2003.

[8] M. Bellare, T. Kohno, and C. Namprempre. Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the Encode-then-Encrypt-and-MAC paradigm. *ACM Transactions on Information and System Security*, 7(2):206–241, May 2004.

[9] M. Bellare, T. Kohno, and C. Namprempre. SSH transport layer encryption modes. IETF RFC 4344, Jan. 2006.

[10] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In T. Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer-Verlag, Berlin Germany, Dec. 2000.

[11] M. Bellare and P. Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography. In T. Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 317–330. Springer-Verlag, Berlin Germany, Dec. 2000.

[12] M. Bellare and P. Rogaway. Code-based game-playing proofs and the security of triple encryption. In S. Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, Lecture Notes in Computer Science. Springer-Verlag, Berlin Germany, 2006.

[13] M. Bellare, P. Rogaway, and D. Wagner. The EAX mode of operation. In B. Roy and W. Meier, editors, *Fast Software Encryption – FSE 2004*, Lecture Notes in Computer Science. Springer-Verlag, Berlin Germany, May 2004.

[14] S. Bellovin. Problem areas for the IP security protocols. In *Proceedings of the 6th USENIX Security Symposium*, pages 1–16, San Jose, California, July 1996.

[15] S. Bellovin and M. Blaze. Cryptographic modes of operation for the internet. In *Second NIST Workshop on Modes of Operation*, 2001.

[16] D. Benedetto, E. Caglioti, and V. Loreto. Language trees and Zipping. *Physical Review Letters*, 88(4), Jan. 2002.

[17] D. Bernstein. Floating-point arithmetic and message authentication, 2000. Available at `http://cr.yp.to/papers.html#hash127`.

[18] D. J. Bernstein. The Poly1305-AES message-authentication code. In H. Gilbert and H. Handschuh, editors, *Fast Software Encryption – FSE 2005*, Lecture Notes in Computer Science. Springer-Verlag, Berlin Germany, 2005.

[19] E. Biham. New types of cryptanalytic attacks using related keys. In T. Helleseth, editor, *Advances in Cryptology – EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 398–409. Springer-Verlag, Berlin Germany, 1993.

[20] E. Biham. How to decrypt or even substitute DES-encrypted messages in 2^{28} steps. *Information Processing Letters*, 84, 2002.

[21] E. Biham and P. Kocher. A known plaintext attack on the PKZIP stream cipher. In B. Preneel, editor, *Fast Software Encryption – FSE ' 94*, volume 1008 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin Germany, 1994.

[22] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway. UMAC: Fast and secure message authentication. In M. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 216–233. Springer-Verlag, Berlin Germany, Aug. 1999.

[23] J. Black and P. Rogaway. A block-cipher mode of operation for parallelizable message authentication. In L. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin Germany, 2002.

[24] N. Borisov, I. Goldberg, and D. Wagner. Intercepting mobile communications: The insecurity of 802.11. In *Seventh Annual International Conference on Mobile Computing and Networking*, 2001.

[25] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In B. Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 451–472. Springer-Verlag, Berlin Germany, 2001.

[26] R. Canetti and H. Krawczyk. Universally composable notions of key exchange and secure channels. In L. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 337–351. Springer-Verlag, Berlin Germany, 2002.

[27] B. Canvel, A. Hiltgen, S. Vaudenay, and M. Vuagnoux. Password interception in a SSL/TLS channel. In D. Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, Lecture Notes in Computer Science. Springer-Verlag, Berlin Germany, 2003.

[28] J. Daemen and V. Rijmen. *The Design of Rijndael: AES–The Advanced Encryption Standard*. Springer-Verlag, Berlin Germany, 2002.

[29] W. Dai. An attack against SSH2 protocol, Feb. 2002. Email to the `ietf-ssh@netbsd.org` email list.

[30] DES modes of operation. National Institute of Standards and Technology, NIST FIPS PUB 81, U.S. Department of Commerce, Dec. 1980.

[31] P. Deutsch. DEFLATE compressed data format specification version 1.3. IETF RFC 1951, May 1996.

[32] W. Diffie and M. E. Hellman. Privacy and authentication: An introduction to cryptography. *Proceedings of the IEEE*, 67(3):397–427, Mar. 1979.

[33] N. Ferguson. Authentication weaknesses in GCM. Public comment to NIST. Available at `http://csrc.nist.gov/CryptoToolkit/modes/comments/CWC-GCM/Ferguson2.pdf`, 2005.

[34] B. Gladman. AES and combined encryption/authentication modes, 2003. Available at `http://fp.gladman.plus.com/AES/index.htm`.

[35] V. Gligor and P. Donescu. Fast encryption and authentication: XCBC encryption and XECB authentication modes. In M. Matsui, editor, *Fast Software Encryption – FSE 2001*, volume 2355 of *Lecture Notes in Computer Science*, pages 92–108. Springer-Verlag, Berlin Germany, 2001.

[36] O. Goldreich, S. Goldwasser, and S. Micali. On the cryptographic applications of random functions. In R. Blakely, editor, *Advances in Cryptology – CRYPTO '84*, volume 196 of *Lecture Notes in Computer Science*, pages 276–288. Springer-Verlag, Berlin Germany, 1985.

[37] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Science*, 28:270–299, 1984.

[38] S. Halevi. EME*: Extending EME to handle arbitrary-length messages with associated data. Cryptology ePrint Archive Report 2004/125, `http://eprint.iacr.org/`, 2004.

[39] C. Hall, I. Goldberg, and B. Schneier. Reaction attacks against several public-key cryptosystems. In V. Varadharajan and Y. Mu, editors, *Proceedings of Information and Communication Security, ICICS'99*, volume 1726 of *Lecture Notes in Computer Science*, pages 2–12. Springer-Verlag, Berlin Germany, Nov. 1999.

[40] Info-ZIP. Info-ZIP note, 20011203, Dec. 2001. Available at `ftp://ftp.info-zip.org/pub/infozip/doc/appnote-011203-iz.zip`.

[41] T. Iwata and K. Kurosawa. OMAC: One-key CBC MAC. In T. Johansson, editor, *Fast Software Encryption – FSE 2003*, Lecture Notes in Computer Science. Springer-Verlag, Berlin Germany, 2003.

[42] K. Jallad, J. Katz, and B. Schneier. Implementation of chosen-ciphertext attacks against PGP and GnuPG. In A. H. Chan and V. D. Gligor, editors, *Information Security, 5th International Conference*, volume 2433 of *Lecture Notes in Computer Science*, pages 90–101. Springer-Verlag, Berlin Germany, 2002.

[43] D. W. Jones. *The Case of the Diebold FTP Site*, July 2003. Available at `http://www.cs.uiowa.edu/~jones/voting/dieboldftp.html`.

[44] C. Jutla. Encryption modes with almost free message integrity. In B. Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 529–544. Springer-Verlag, Berlin Germany, May 2001.

[45] B. Kaliski. PKCS #5: Password-based cryptography specification version 2.0. IETF RFC 2898, Sept. 2000.

[46] J. Katz and B. Schneier. A chosen ciphertext attack against several e-mail encryption protocols. In *Ninth USENIX Security Symposium*, 2000.

[47] J. Katz and M. Yung. Unforgeable encryption and chosen ciphertext secure modes of operation. In B. Schneier, editor, *Fast Software Encryption – FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 284–299. Springer-Verlag, Berlin Germany, Apr. 2000.

[48] J. Kelsey. Compression and information leakage of plaintext. In V. Rijmen and J. Daemen, editors, *Fast Software Encryption – FSE 2002*, volume 2365 of *Lecture Notes in Computer Science*, pages 263–276. Springer-Verlag, Berlin Germany, 2002.

[49] T. Kohno. Attacking and repairing the WinZip encryption scheme. In B. Pfitzmann, editor, *Proceedings of the 11th ACM Conference on Computer and Communications Security*, pages 72–81. ACM Press, Oct. 2004.

[50] T. Kohno, J. Viega, and D. Whiting. CWC: A high-performance conventional authenticated encryption mode. In B. Roy and W. Meier, editors, *Fast Software Encryption*, volume 3017 of *Lecture Notes in Computer Science*, pages 408–426. Springer-Verlag, Feb. 2004.

[51] H. Krawczyk. LFSR-based hashing and authentication. In Y. Desmedt, editor, *Advances in Cryptology – CRYPTO '94*, Lecture Notes in Computer Science. Springer-Verlag, Berlin Germany, Aug. 1994.

[52] H. Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is SSL?). In J. Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 310–331. Springer-Verlag, Berlin Germany, Aug. 2001.

[53] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-hashing for message authenticationa. IETF Internet Request for Comments 2104, Feb. 1997.

[54] H. Lipmaa. AES/Rijndael: speed, 2003. Available at `http://www.tcs.hut.fi/~helger/aes/rijndael.html`.

[55] H. Lipmaa, P. Rogaway, and D. Wagner. CTR-mode encryption. In *First NIST Workshop on Modes of Operation*, 2000.

[56] M. Luby and C. Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Computation*, 17(2), Apr. 1988.

[57] D. McGrew. Integer counter mode, Oct. 2002. Available at `http://www.ietf.org/internet-drafts/draft-irtf-cfrg-icm-01.txt`.

[58] D. McGrew. The truncated multi-modular hash function (TMMH), version two, Oct. 2002. Available at `http://www.ietf.org/internet-drafts/draft-irtf-cfrg-tmmh-00.txt`.

[59] D. McGrew. The universal security transform, Oct. 2002. Available at `http://www.ietf.org/internet-drafts/draft-irtf-cfrg-ust-01.txt`.

[60] D. McGrew and J. Viega. The security and performance of the Galois/Counter Mode (GCM) of operation. In A. Canteaut and K. Viswanathan, editors, *Progress in Cryptology – INDOCRYPT 2004*, volume 3348 of *Lecture Notes in Computer Science*, pages 343–355. Springer-Verlag, Berlin Germany, Dec. 2004.

[61] I. Mironov. (Not so) random shuffles of RC4. In M. Yung, editor, *Advances in Cryptology – CRYPTO 2002*, Lecture Notes in Computer Science, pages 304–319. Springer-Verlag, Berlin Germany, 2002.

[62] C. Namprempre. Secure channels based on authenticated encryption schemes: A simple characterization. In Y. Zheng, editor, *Advances in Cryptology – ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 515–532. Springer-Verlag, Berlin Germany, Dec. 2002.

[63] M. Naor and O. Reingold. On the construction of pseudorandom permutations: Luby-rackoff revisited. *J. Cryptology*, 12(1):29–66, 1999.

[64] W. Nevelsteen and B. Preneel. Software performance of universal hash functions. In J. Stern, editor, *Advances in Cryptology – EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 24–41. Springer-Verlag, Berlin Germany, 1999.

[65] PKWARE. APPNOTE.TXT - .ZIP File Format Specification, Apr. 2004. Version 6.2.0, available at `http://www.pkware.com/products/enterprise/white_papers/appnote.txt`.

[66] PKWARE. APPNOTE.TXT - .ZIP File Format Specification, Jan. 2004. Version 6.1.0, replaced by [65].

[67] P. Rogaway. Problems with proposed IP cryptography, 1995. Available at `http://www.cs.ucdavis.edu/~rogaway/papers/draft-rogaway-ipsec-comments-00.txt`.

[68] P. Rogaway. Bucket hashing and its applications to fast message authentication. *Journal of Cryptology*, 12:91–115, 1999.

[69] P. Rogaway. Authenticated encryption with associated data. In V. Atluri, editor, *Proceedings of the 9th Conference on Computer and Communications Security*. ACM Press, Nov. 2002.

[70] P. Rogaway. The AEM authenticated-encryption mode, 2003. Specification 1.3, available at `http://www.cs.ucdavis.edu/~rogaway/papers/offsets.html`.

[71] P. Rogaway. Nonce-based symmetric encryption. In B. Roy and W. Meier, editors, *Fast Software Encryption – FSE 2004*, Lecture Notes in Computer Science. Springer-Verlag, Berlin Germany, May 2004.

[72] P. Rogaway, M. Bellare, and J. Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Transactions on Information and System Security*, 6(3):365–403, 2003.

[73] P. Rogaway and D. Wagner. A critique of CCM. Cryptology ePrint Archive Report 2003/070, `http://eprint.iacr.org/`, 2003.

[74] V. Shoup. On fast and provably secure message authentication based on universal hashing. In N. Koblitz, editor, *Advances in Cryptology – CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 313–328. Springer-Verlag, Berlin Germany, Aug. 1996.

[75] V. Shoup. Sequences of games: A tool for taming complexity in security proofs. Cryptology ePrint Archive Report 2004/332, `http://eprint.iacr.org/`, 2004.

[76] D. X. Song, D. Wagner, and X. Tian. Timing analysis of keystrokes and timing attacks on SSH. In *Proceedings of the 10th USENIX Security Symposium*, pages 337–352, Washington, DC, Aug. 2001.

[77] M. Stay. ZIP attacks with reduced known plaintext. In M. Matsui, editor, *Fast Software Encryption – FSE 2001*, volume 2355 of *Lecture Notes in Computer Science*, pages 124–134. Springer-Verlag, Berlin Germany, 2001.

[78] D. Stinson. Universal hashing and authentication codes. *Designs, Codes and Cryptography*, 4:369–380, 1994.

[79] S. Vaudenay. Security flaws induced by CBC padding – applications to SSL, IPSEC, WTLS .... In L. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 534–545. Springer-Verlag, Berlin Germany, 2002.

[80] D. Wagner and B. Schneier. Analysis of the SSL 3.0 protocol. In *Proceedings of the Second USENIX Workshop on Electronic Commerce*, 1996.

[81] M. Wegman and L. Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22:265–279, 1981.

[82] D. Whiting, N. Ferguson, and R. Housley. Counter with CBC-MAC (CCM). Submission to NIST. Available at http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/, 2002.

[83] WinZip Computing, Inc. AES encryption information: Encryption specification AE-2, Jan. 2004. Version 1.02, available at http://www.winzip.com/aes_info.htm.

[84] WinZip Computing, Inc. Download WinZip add-ons, Apr. 2004. Available at http://www.winzip.com/daddons.htm.

[85] WinZip Computing, Inc. Homepage, Mar. 2004. Available at http://www.winzip.com/.

[86] WinZip Computing, Inc. What's new in WinZip 9.0, Mar. 2004. Available at http://www.winzip.com/whatsnew90.htm.

[87] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen. SSH transport layer protocol, 2002. Draft 12, available at http://www.ietf.org/html.charters/secsh-charter.html.