Real-World Aspects of Secure Channels: Fragmentation, Causality, and Forward Security

Vom Fachbereich Informatik der
Technischen Universität Darmstadt genehmigte

Dissertation

zur Erlangung des Grades
Doctor rerum naturalium (Dr. rer. nat.)
von

Giorgia Azzurra Marson, M.Sc.
geboren in Rom

Referenten: Prof. Dr. Marc Fischlin
Dr. Martijn Stam

Tag der Einreichung: 30.08.2016
Tag der mündlichen Prüfung: 12.10.2016

Darmstadt, 2017
D 17

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit – abgesehen von den in ihr ausdrücklich genannten Hilfen – selbständig verfasst habe.

Giorgia Azzurra Marson

Academic CV

October 2006 – December 2009
Bachelor of Science in Mathematics, Sapienza University of Rome

January 2010 – December 2011
Master of Science in Applied Mathematics, Sapienza University of Rome

January 2012 – October 2016
PhD candidate in Computer Science, Technische Universität Darmstadt

List of Publications

[1] Giorgia Azzurra Marson and Bertram Poettering. Security Notions for Bidirectional Channels. *IACR Transactions on Symmetric Cryptology*, 2017. (To appear.) **Part of this thesis**.

[2] Jean Paul Degabriele, Victoria Fehr, Marc Fischlin, Tommaso Gagliardoni, Felix Günther, Giorgia Azzurra Marson, Arno Mittelbach, and Kenneth G. Paterson. Unpicking PLAID: a cryptographic analysis of an iso-standards-track authentication protocol. *Int. J. Inf. Sec.*, 15(6):637–657, 2016.

[3] Sedat Akleylek, Nina Bindel, Johannes Buchmann, Juliane Krämer, and Giorgia Azzurra Marson. An Efficient Lattice-Based Signature Scheme with Provably Secure Instantiation. In David Pointcheval, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *Progress in Cryptology - AFRICACRYPT 2016 - 8th International Conference on Cryptology in Africa, Fes, Morocco, April 13-15, 2016, Proceedings*, volume 9646 of *Lecture Notes in Computer Science*, pages 44–60. Springer, 2016.

[4] Marc Fischlin, Felix Günther, Giorgia Azzurra Marson, and Kenneth G. Paterson. Data Is a Stream: Security of Stream-Based Channels. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 545–564. Springer, 2015. **Part of this thesis**.

[5] Jean Paul Degabriele, Victoria Fehr, Marc Fischlin, Tommaso Gagliardoni, Felix Günther, Giorgia Azzurra Marson, Arno Mittelbach, and Kenneth G. Paterson. Unpicking PLAID - A Cryptographic Analysis of an ISO-standards-track Authentication Protocol. In Liqun Chen and Chris Mitchell, editors, *1st International Conference on Research in Security Standardisation, London, U.K., December 16-17, 2014. Proceedings*, volume 8893 of *Lecture Notes in Computer Science*, pages 1–25. Springer, 2014.

[6] Giorgia Azzurra Marson and Bertram Poettering. Even More Practical Secure Logging: Tree-Based Seekable Sequential Key Generators. In Miroslaw Kutylowski and Jaideep Vaidya, editors, *Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wrocław, Poland, September 7-11, 2014. Proceedings, Part II*, volume 8713 of *Lecture Notes in Computer Science*, pages 37–54. Springer, 2014. **Part of this thesis**.

[7] Giorgia Azzurra Marson and Bertram Poettering. Practical Secure Logging: Seekable Sequential Key Generators. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings*, volume 8134 of *Lecture Notes in Computer Science*, pages 111–128. Springer, 2013. **Part of this thesis**.

[8] Özgür Dagdelen, Marc Fischlin, Tommaso Gagliardoni, Giorgia Azzurra Marson, Arno Mittelbach, and Cristina Onete. A Cryptographic Analysis of OPACITY - (Extended Abstract). In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings*, volume 8134 of *Lecture Notes in Computer Science*, pages 345–362. Springer, 2013.

[9] Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the Non-malleability of the Fiat–Shamir Transform. In Steven D. Galbraith and Mridul Nandi, editors, *Progress in Cryptology - INDOCRYPT 2012, 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings*, volume 7668 of *Lecture Notes in Computer Science*, pages 60–79. Springer, 2012.

[10] Giorgia Azzurra Marson and Bertram Poettering. Causality Matters: Security Models and Constructions for Bidirectional and Broadcast Channels. **Unpublished manuscript, Part of this thesis**.

Acknowledgments

During my PhD time several people contributed to my development as a scientist and as a person. They all, directly or indirectly, had an influence on this work.

I was very fortunate to have the opportunity of joining Marc Fischlin's team *Cryptoplexity*. Marc shared with me relevant research questions from the start. He guided me with steady support and a lot of patience, especially in the first phase of my PhD, when I was looking for my specific direction in research. He always found the time to discuss about cryptography and anything else I needed his advise for. He encouraged me to visit conferences, workshops, and schools all over the world. I am deeply thankful to him for the time and care he dedicated to me—and he continues to dedicate to his amazing group.

The results presented in this thesis are the product of engaging collaborations with Marc Fischlin, Felix Günther, Kenny Paterson, and Bertram Poettering.

Doing research with Felix was a mix of looking at problems from orthogonal angles and meeting each other with similar solutions and interpretations. Working with him has been constructive, refreshing, and a lot of fun. Beyond being a fantastic colleague and co-author, Felix is also great as a friend and as a travel companion. I will always be grateful for the unforgettable time we spent together, in Darmstadt and around the world.

Doing research with Bertram was somewhat the opposite: tackling problems from the same angle and then diverging on how to solve them (until we finally had an agreement). Bertram and I had countless research discussions,face to face and remotely, at any time of the day. Some of these discussions ended up in joint projects. All of them contributed largely to what I learned about cryptography. I thank him for all this, but also for the nice time we jointly had in Darmstadt, Egham, Bochum, and during our numerous trips.

I am deeply grateful to Kenny for the incredible amount of time and focused thought he offered to Felix and me during a research visit at Royal Holloway in January 2015. It has been also inspiring, and a great pleasure, to chat with Kenny during coffee breaks at conferences.

During my PhD I had the chance to work with other researchers. In particular I thank Nina Bindel, Özgür Dagdelen, Jean Paul Degabriele, Victoria Fehr, Tommaso Gagliardoni, Juliane Krämer, Arno Mittelbach, and Cristina Onete for the engaged collaborative work.

Darmstadt is like a second home for me. There I met fantastic people that are—and will be—deeply connected to me. I thank (in alphabetical order) Paul Baecher, Nina Bindel, Jacqueline Brendel, Chris Brzuska, Özgür (Özi) Dagdelen, Pooya Farshim, Victoria Fehr, Tommaso Gagliardoni, Felix and Juliane Günther, Christian Janson, Agnes Kiss, Juliane Krämer, Sogol Mazaheri, Heike Meißner, Arno Mittelbach, Cristina Onete, Andreas Peter, Bertram Poettering, Andrea Püchner, and Cristian Staicu for making my time in Darmstadt so enjoyable. I thank in particular Özi, for taking care of me even before I moved to Darmstadt and keeping always an eye on me; Pooya, for encouraging me, challenging me, and sharing his thoughts about research as well as other relevant matters; and Chris, for all the support and care.

I am grateful to Martijn Stam for agreeing to be the co-referee of this thesis and for valuable comments and discussions, and to Patrick Eugster, Iryna Gurevych, and Matthias Hollick for

joining the committee of my defense.

Last but not least, I am deeply thankful to my brother Alessio and my parents for their love and support.

Giorgia Azzurra Marson

My Contribution

The results presented in this thesis are the outcome of inspiring discussions and engaging collaborative work with Marc Fischlin, Felix Günther, Kenny Paterson, and Bertram Poettering. I would like to thank them all for their contribution to our joint work. As it is common in a collaborative research project, each individual achievement may be contributed by several, and most often all, researchers participating to that project through sharing new ideas, questioning results, discussing strategies, formalizing one's intuition, etc. This makes it hard, if not impossible, to pinpoint who contributed which specific part of the overall work. In what follows I will give an account of the results presented in this thesis and pinpoint, when possible, what my specific contribution is.

The results of Chapter 3 are joint work with Marc Fischlin, Felix Günther, and Kenny Paterson. Felix and I developed together functionality, confidentiality and integrity notions for stream-based channels (presented in Sections 3.2 and 3.3). Concerning the other results of Chapter 3, I focused on studying the relations among confidentiality and integrity for stream-based channels (presented in Section 3.4), in particular, on proving that the classic result of [BN00] asserting that confidentiality against passive adversaries together with integrity of ciphertexts implies confidentiality against active adversaries also holds in the streaming setting, as long as decryption errors are predictable. This result is stated in Theorem 1. Felix instead focused on the construction presented in Section 3.5, which builds a stream-based channel generically from AEAD. He worked out the security proofs and explained to what extent the TLS record protocol design compares to our construction.

The results of Chapters 7 and 8 are based on joint work with Bertram Poettering that appeared in [MP13, MP14]. In the conference papers we left open whether our security model for sequential key generators is strictly stronger than the one presented in [BY03]. As I prove in this thesis, the two models turn out to be equivalent. This result does not appear elsewhere. The notion of shortcut one-way permutations and the security proofs of our SSKG constructions (presented in Sections 8.3 and 8.4) is my work. Section 8.5 reproduces the experimental results appearing in [MP13] that compare our SSKG constructions, based on pseudorandom generators and on shortcut one-way permutations, respectively. Both the scheme's implementation and the performance analysis is Bertram's contribution.

The results of Chapters 4, 5 and 6 are also joint work with Bertram Poettering. The security analysis of the canonic composition of unidirectional channels (Sections 5.5 and 5.6.2) is exclusively my contribution; it appears in [MP17]. The other results presented in this thesis were developed jointly and do not appear elsewhere.

Abstract

A secure channel is a cryptographic protocol that adds security to unprotected network connections. Prominent examples include the Transport Layer Security (TLS) and the Secure Shell (SSH) protocols. Because of their large-scale deployment, these protocols received a lot of attention from academia. Starting with the seminal work of Bellare, Kohno, and Namprempre (BKN; CCS 2002) on the security of SSH, numerous authors analyzed channel protocols using the same approach of BKN to model a channel as a stateful authenticated encryption scheme. However, deployed protocols such as TLS and SSH are inherently complex, and a single mathematical abstraction can hardly capture all aspects that are relevant to security.

In this thesis we reconsider the suitability of the stateful authenticated encryption abstraction for the analysis of real-world channel protocols. In particular, we highlight that such an abstraction is too restrictive, in a sense that we clarify next, to capture three important aspects that do not appear in existing cryptographic models for secure channels.

Firstly, we question the common approach that treats secure channels using atomic encryption and decryption interfaces to transport a sequence of messages. This approach ignores that many real-world protocols, including TLS and SSH, offer a streaming interface instead. To formalize the non-atomic behavior of these protocols we initiate the study of stream-based channels and their security. We formalize notions of confidentiality and integrity by extending the BKN model for stateful authenticated encryption to take the peculiarities of streams into account. Inspired by the TLS 1.3 protocol we present a generic construction of a stream-based channel from any authenticated encryption scheme with associated data (AEAD), and prove its security.

Secondly, we note that while TLS, SSH, and many other channel protocols are typically used for bidirectional interaction, cryptographic models assessing the security of these protocols exclusively account for unidirectional communication, from one sender to one receiver. We correspondingly ask: Do security results for unidirectional channels extend to the bidirectional case? And, in the first place, what does security in the bidirectional setting actually mean? How does all this scale when more than two participants are involved? To answer these questions we conduct a rigorous study of security notions for bidirectional channels and their generalization to the broadcast setting with more than two participants. In a broadcast scenario, confidentiality and integrity need to capture aspects related to the causality of events in distributed systems that have no counterpart in the much simpler unidirectional case. The causality between exchanged messages is particularly relevant, both in terms of functionality and of security, in the context of instant messaging protocols such as TextSecure. Furthermore, we provide generic constructions of broadcast channels from AEAD. We also analyze and validate a traditional heuristic (used, among others, in TLS) of combining two unidirectional channels to realize a bidirectional one.

Finally, we look at forward security, which strengthens regular security by demanding that even if an adversary eventually obtains the secret key in use (by corruption), past uses of the cryptosystem are not compromised. While being a standard requirement for authenticated

key exchange, so far, providing forward security was not considered a goal of cryptographic channels in the BKN model and its follow-ups. However, it is considered folklore that forward-secure authenticated encryption schemes can be constructed in a modular way by replacing the key generation procedure with one that produces a sequence of forward-secure keys and then using these keys for re-keying the encryption and decryption routines in use. This approach may also be applied for realizing forward-secure channels. Following this idea, in the last part of the thesis we leave the domain of channels and focus on building forward-secure key generation mechanisms. Secure solutions for the latter primitive have been proposed already in the past. In this thesis we complement the current picture by contributing certain efficiency improvements for sequential key generators. In particular, we illustrate that our solutions find a natural application in the authentication of log files. Implementations of one of our schemes are installed on millions of computers world-wide.

Zusammenfassung

Ein sicherer Kanal ist ein kryptographisches Protokoll das ungeschützte Netzwerkverbindungen absichert. Die prominentesten Beispiele, *Transport Layer Security* (TLS) und *Secure Shell* (SSH), erhielten aufgrund ihrer weiten Verbreitung besondere Aufmerksamkeit von Seiten akademischer Forschung. Beginnend mit den Arbeiten von Bellare, Kohno, und Namprempre (BKN; CCS 2002) über die Sicherheit von SSH haben zahlreiche Autoren kryptographische Kanalprotokolle mit einem Ansatz ähnlich dem von BKN analysiert, nämlich Kanäle als zustandsbehaftete authentisierte Verschlüsselung modellierend. Verbreitete Protokolle wie TLS und SSH sind jedoch ihrem Wesen nach komplex, und eine einzelne mathematische Abstraktion kann schwerlich alle für ihre Sicherheit relevanten Aspekte berücksichtigen.

In dieser Dissertation stellen wir die Eignung der Abstraktion als zustandsbehaftete authentisierte Verschlüsselung für die Analyse von Kanalprotokollen in der realen Welt in Frage. Insbesondere zeigen wir auf, dass eine solche Abstraktion zu restriktiv ist, auf eine Weise die wir im Folgenden erklären, um drei wichtige Aspekte abzudecken, die in bisher existierenden kryptographischen Modellierungen von Kanälen nicht auftreten.

Zunächst stellen wir den allgegenwärtigen Ansatz in Frage, der Kanäle aufgrund entsprechend formalisierter Schnittstellen als Transportmittel für atomare Nachrichten auffasst. Der genannte Ansatz ignoriert, dass viele Protokolle in der realen Welt, einschließlich TLS und SSH, tatsächlich strom-orientierte Schnittstellen anbieten. Zum Studieren des nicht-atomaren Verhaltens solcher Protokolle und ihrer Sicherheit entwickeln wir Vertraulichkeits- und Integritätsbegriffe auf Basis des Modells für zustandsbehaftete authentisierte Verschlüsselung von BKN, um den Eigenheiten von strom-basierten Protokollen Rechnung zu tragen. Inspiriert vom TLS 1.3-Protokoll präsentieren wir eine generische Konstruktion eines strom-basierten Kanals, auf Grundlage von authentisierter Verschlüsselung mit assoziierten Daten (AEAD), und beweisen ihre Sicherheit.

Zweitens stellen wir fest, dass TLS, SSH, und viele weitere Kanalprotokolle typischerweise für bidirektionale Interaktion verwendet werden, während kryptographische Modelle solche Protokolle ausschließlich im Hinblick auf unidirektionale Kommunikation bewerten, d.h., mit genau einem Sender und einem Empfänger. Entsprechend fragen wir: Lassen sich Ergebnisse über unidirektionale Kanäle auf den bidirektionalen Fall übertragen? Und, unmittelbarer, was bedeutet Sicherheit im bidirektionalen Fall überhaupt? Wie skalieren diese Eigenschaften wenn mehr als zwei Teilnehmer vorhanden sind? Um diese Fragen zu beantworten studieren wir Sicherheitsbegriffe für bidirektionale Kanäle und ihre Verallgemeinerung zum *Broadcast Setting*, in dem mehrere Teilnehmer vorhanden sind und jede Sendung eines Teilnehmers alle übrigen Teilnehmer erreicht. Im letzteren Fall müssen die Definitionen von Vertraulichkeit und Integrität Aspekte von Kausalität in Verteilten Systemen umfassen, die keine Entsprechung im viel einfacheren Fall unidirektionaler Kommunikation haben. Kausale Zusammenhänge zwischen ausgetauschten Nachrichten sind besonders relevant, in Hinblick auf sowohl Funktionalität als auch Sicherheit, im Kontext von Protokollen zum *Instant Messaging*, etwa TextSecure. In diesem Teil der Dissertation geben wir generische Konstruktionen von *Broadcast*-Kanälen auf Basis von

AEAD an. Wir analysieren und validieren die traditionelle heuristische Methode (benutzt etwa für TLS), einen bidirektionalen Kanal durch die Kombination zweier unidirektionaler Kanäle zu erhalten.

Im dritten Teil dieser Arbeit studieren wir Vorwärtssicherheit, die reguläre Sicherheitsdefinitionen durch die Forderung stärkt, dass sogar im Fall wenn Angreifer (durch Korrumpierung) zu einem Zeitpunkt Zugriff auf verwendetes Schlüsselmaterial erhalten, vergangene Anwendungen des Kryptosystems nicht kompromittiert werden. Obwohl dies ein standardmäßiges Sicherheitsziel von authentisiertem Schlüsselaustausch ist, wurde sein Erreichen bisher nicht als ein Ziel von kryptographischen Kanälen verstanden, zumindest nicht nach den Modellen von BKN und seinen Nachfolgern. Es wird jedoch als Folklore-Ergebnis aufgefasst, dass vorwärtssichere authentisierte Verschlüsselung auf modulare Weise erreicht werden kann durch Ersetzen der Schlüsselerzeugungsprozedur durch eine, die eine Folge von vorwärtssicheren Schlüsseln erzeugt, um dann diese zum Erneuern der Schlüssel von Ver- und Entschlüsselungsroutine zu verwenden. Dieser Ansatz kann auch zum Realisieren vorwärtssicherer Kanäle verwendet werden. Dem folgend verlassen wir im dritten Teil der Dissertation das Gebiet sicherer Kanäle und konzentrieren uns auf die Konstruktion vorwärtssicherer Schlüsselerzeugungsmechanismen. Sichere Lösungen bereits in der Vergangenheit vorgeschlagen worden. Hier komplementieren wir das aktuelle Bild durch explizit effizienzsteigernde Verbesserungen für vorwärtssicheren Schlüsselerzeugungsmechanismen, und illustrieren, dass unsere Lösungen insbesondere eine natürliche Anwendung in der Authentisierung von Log-Dateien finden. Implementationen einer unserer Konstruktionen sind auf Millionen von Computern weltweit installiert.

Contents

1 Introduction **1**

2 Notation and Definitions **7**
2.1 General Notation 7
2.2 Cryptographic Assumptions and Primitives 8
2.3 Cryptographic Models for Secure Channels 10

3 Stream-Based Channels **17**
3.1 Introduction 17
3.2 Syntax and Functionality 18
3.3 Defining Security for Stream-Based Channels 21
3.4 Relations Among Notions 29
3.5 Constructions 34
3.6 A Note on the TLS Record Protocol 39

4 Broadcast Communication **41**
4.1 Introduction 41
4.2 Communication Graphs 42
4.3 Technical Results 47

5 FIFO Channels **53**
5.1 Introduction 53
5.2 Syntax and Functionality 54
5.3 Defining Security for FIFO Channels 56
5.4 Relations Among Notions 59
5.5 Unidirectional Channels 61
5.6 Constructions 64

6 Causal Channels **73**
6.1 Introduction 73
6.2 Syntax and Functionality 75
6.3 Security and Relations Among Notions 76
6.4 Constructions 78

7 Sequential Key Generators **87**
7.1 Introduction 87
7.2 Syntax and Functionality 88
7.3 Security 89
7.4 Comparison with Stateful Generators 90
7.5 Constructions 93

7.6 A Digression on Secured Local Logging 94

8 Seekable Sequential Key Generators **95**
8.1 Introduction 95
8.2 Seekability 95
8.3 Shortcut One-Way Permutations 97
8.4 Constructions 99
8.5 Practical aspects and deployment 111

9 Conclusion and Open Problems **115**

Bibliography **117**

Chapter 1

Introduction

Cryptography originates from the need to communicate securely. Modern cryptography goes hand in hand with provable security, which offers a mathematical framework to determine whether and in how far a given cryptosystem is secure. The provable security approach essentially reduces to the following steps. Firstly, one defines formally what security means for a given cryptographic task. A security definition consists of a mathematical model that describes the expected functionality of cryptosystems that achieve the targeted task as well as how these systems should behave if an attacker misuses them. Secondly, one proves that a specific cryptosystem is secure by presenting a reduction, i.e., an efficient algorithm turning any adversary that breaks the security of the cryptosystem (according to the definition of security established in the first step) into one that solves a problem that is assumed to be hard. Then, if the chosen problem is truly hard, security of the cryptosystem is implied.

In fact, a proof of security is a conditional result: one can only derive the conclusion that security holds in an idealized world where the underlying hardness assumption is true and the scheme in place can be (mis)used by attackers only to the extent described by the model. Thus, for sound security results it is extremely important to rely on well-studied assumptions and to make the security model as close as possible to the reality. The focus of this thesis will be on modeling security.

Secure Channels in Cryptography

The fundamental applications of cryptography—critical for nearly every operation in our interconnected world—remains secure communication. Suppose that two parties, Alice and Bob, wish to communicate in a secure way over the Internet. In an ideal world with no adversaries they would simply do so by exchanging messages in the clear over a reliable network, for instance over TCP/IP. In the real world, however, adversaries exist who may eavesdrop on the communication or manipulate the transmitted data. To protect against this threat, Alice and Bob can strengthen their network connection by adding a layer of protection: they may use a cryptographic channel (a.k.a. secure channel). Widely-deployed examples of cryptographic channels include the Transport Layer Security (TLS) [DR08] and the Secure Shell (SSH) [YL06] protocols, the encryption layer of mobile telecommunication systems, and protocols connecting bank computers with ATMs, to name a few.

Informally, a secure channel is similar to a symmetric encryption scheme: assuming that Alice and Bob share some key material, Alice can encapsulate a message into a ciphertext, send the latter to Bob over the (unprotected) network, and be sure that only Bob will be able to decrypt the ciphertext; this property is called *confidentiality*. Moreover, Bob can be sure

that if he receives a ciphertext that was not genuinely produced by Alice then the channel will explicitly notify him that an adversarial manipulation took place; this property is called *integrity*. The cryptographic tool that simultaneously offers both confidentiality and integrity (in the symmetric-key setting) is *authenticated encryption*.

Authenticated encryption (AE) has emerged as being a natural cryptographic tool for building secure channels; however, it does not constitute a secure channel on its own. For instance, in most practical situations a secure channel should also guarantee detection of (and possibly recovery from) out-of-order delivery and replays of ciphertexts. As another difference, some secure channel designs have additional features that can be used to provide protection against traffic analysis. Furthermore, a secure channel should deal with error handling; it may accept messages of arbitrary length and, for technical reasons, fragment these before encryption, and may reassemble these fragments again after decryption; alternatively, it may present to applications a maximum message size that is well-matched to the underlying network.

A rigorous study of channel security was initiated by Bellare, Kohno, and Namprempre (BKN) [BKN02] to assess the security of the SSH Binary Packet Protocol. These authors proposed as an abstraction of a secure channel the notion of *stateful authenticated encryption*, which extends the standard notions of confidentiality and integrity to also incorporate protection against replays and reorderings. Later work by the same and other authors also used the stateful AE primitive to analyze real-world channel protocols. For instance, [JKSS12] and [KPW13] model the TLS channel protocol as a stateful AE scheme that additionally accepts associated data (AEAD) [Rog02] and specifically aims at hiding the length of encrypted plaintexts.

The analysis of Bellare *et al.* [BKN02] provably shows that the SSH Binary Packet Protocol achieves security in the stateful AE sense. However, a few years after the analysis was published, Albrecht *et al.* [APW09] developed a plaintext-recovery attack on SSH, questioning the confidentiality of SSH. There is no flaw in the proof of Bellare *et al.* [BKN02], though. The attack of [APW09] specifically exploits the adversary's ability to deliver arbitrary sequences of SSH packet fragments to the receiver and observe the receiver's behavior upon processing each of the fragments. In contrast, the BKN model assumes that the adversary can only see the reaction of the decryption algorithm on input atomic, rather than fragmented, ciphertexts. With other words, the attack presented by Albrecht *et al.* cannot be expressed within the BKN model. A similar issue concerns the CBC-mode based construction of MAC-then-Encrypt (MtE) of TLS, which enjoys a proof of security by Krawczyk [Kra01] while its OpenSSL implementation was later shown to be vulnerable to a plaintext-recovery attack by Canvel *et al.* [CHVV03]. Here the attack was possible because the targeted implementation leaks on decrypting invalid ciphertexts more than the mere fact that decryption failed, while the security model of [Kra01], instead, assumes that the decryption algorithm always outputs the same error symbol, thus only revealing that decryption failed.

The above attacks highlight the importance (and the difficulty) of designing security models that adequately match how protocols can be attacked in reality. In order to model the plaintext recovery attack given by Albrecht *et al.* [APW09] against the SSH channel protocol, Boldyreva, Degabriele, Paterson, and Stam (BDPS) [BDPS12] proposed an extension of symmetric encryption that allows the decryption algorithm to accept fragmented ciphertexts, and developed corresponding confidentiality notions. The same authors in [BDPS14] revised syntax and security of (stateless and stateful) authenticated encryption by letting the decryption algorithm output distinguishable decryption errors, again to make attacks like the one by Canvel *et al.* [CHVV03] visible.

Stream-Based Channels

Characteristic of all the above-mentioned prior works on channels is that they treat secure channels as providing an *atomic* interface for messages, meaning that the channel is designed only for sending sequences of messages that are considered units. However, this only captures a fraction of secure channel designs that are actually used in practice. For example, even though the TLS specification does not include a formal API definition, it is clear that the design intention is to provide a secure channel for data streams. A stream-oriented treatment of channels is neither possible in the traditional AE models nor in the model proposed by Boldyreva *et al.* [BDPS12] that allows for ciphertext fragmentation.

To capture this important aspect, in Chapter 3 we develop a new channel primitive that we call a *stream-based channel*. We provide functional specifications and corresponding confidentiality and integrity notions for this channel type. Differently from traditional (stateful) encryption primitives, a stream-based channel is expected to provide secure transmission of a stream of data rather than a sequence of atomic messages. These syntactical differences are reflected in the security notions, that become considerably more complex than the security notions for stateful AE. While our methodology and modeling closely resembles that of Boldyreva *et al.* [BDPS12], and indeed builds upon this work, a crucial difference lies in our treatment of the sending algorithm, which is atomic in [BDPS12] while it allows for fragmentation in the case of stream-based channels and can arbitrarily buffer and fragment the input message when preparing ciphertexts for sending. As a consequence, while for symmetric encryption supporting fragmentation there is a one-to-one correspondence between sent messages and sent ciphertexts, in stream-based channels such a correspondence is lost. This requires careful reconsideration of the confidentiality definitions of [BDPS12]. In addition, we develop suitable integrity notions for the streaming setting, whereas [BDPS12] only defines confidentiality. Bringing integrity into the picture for stream-based channels also enables us to prove a composition result analogous to the classical result of [BN00] for symmetric encryption, which states that confidentiality against passive adversaries in combination with integrity of ciphertexts guarantees confidentiality against active adversaries. To demonstrate the feasibility of our notions we provide a construction of a stream-based channel that uses AEAD as a component and achieves the strongest confidentiality and integrity notions that we define. This construction closely mimics the use of AEAD in the TLS Record Protocol; our results thus serves as a validation of the latter.

Bidirectional and Broadcast Channels

All the above security models for channel protocols assume that a sender transmits data to a receiver, providing an abstraction of a *unidirectional* channel. However, secure channels are typically used for bidirectional interaction instead. Concretely, bidirectional channels are often realized by running two unidirectional channels in opposite directions. For instance, in TLS the handshake protocol establishes a total of four keys: two are used to protect the communication in one direction, the other two for protecting the other direction. Thus, security claims about TLS and SSH like those in [BKN02, PRS11, JKSS12, KPW13] (which rely on stateful AE abstractions), in fact, establish results for the unidirectional components of the TLS and SSH channel protocols. A natural question arises: if TLS and SSH are secure in a unidirectional sense, are they also secure *bidirectional* channels? One may naively argue that they are, as long as their components provide unidirectional security. However, cryptographic research shows that, in general, using secure building blocks does not necessarily imply that the resulting composed protocol is secure, too. Even worse, what security means in the bidirectional case

is not even defined (yet). Thus, whether TLS and SSH are secure as bidirectional channels remains, so far, an open issue.

In this thesis we develop appropriate security notions for bidirectional channels. We further lift them to the scenario of multiple parties that communicate in a broadcast fashion, i.e., such that each message is transmitted to all communicating parties. In both cases the interactiveness introduces new challenges that have no counterpart in the much simpler unidirectional setting. As a basis of our considerations, in Chapter 4 we formalize different models of network communication, in particular those of FIFO and causal broadcast (bidirectional communication is a special case of broadcast). FIFO broadcast is a natural extension of (two-party) unidirectional communication: it guarantees that the messages sent by each participant are delivered reliably to all other participants, i.e., following the first-in-first-out (FIFO) principle. Causal broadcast provides stronger guarantees than FIFO delivery: a participant should receive a message only if he received all messages that were globally sent before (where the term 'globally' emphasizes that we do not restrict the attention to any specific sender but consider a global ordering of sending and receiving events). This property is particularly suitable for applications in which more than two participants wish to communicate simultaneously, e.g., in a multi-party chat.

In Chapters 5 and 6 we introduce notions of cryptographic channels that run on top of these networks, FIFO and causal channels, and study their security. In lifting the common definitions of confidentiality and integrity to the multi-user and multi-directed setting, we incorporate the ordering properties into the security definitions. For instance, in causal channels, if an adversary does not modify any ciphertext but instead violates the causal order of deliveries, also this should be considered an active attack. According to our notions, causal channels come with an extended functionality: we require that, upon each invocation, sending and receiving algorithms additionally output a history of the past communication which informs the user about the causal relationships among all the messages that he has seen so far. We formalize a corresponding integrity notion that prevents adversaries from falsifying this history output without detection. We formally assess the relations among the new security notions and, beyond others, confirm the expectation that also in the multi-directed setting confidentiality against passive adversaries together with strong integrity implies confidentiality against active adversaries (this is analogous to the result first proven in [BN00] for authenticated encryption), both in the FIFO and in the causal setting. As a feasibility result we show how to build generically a FIFO channel from AEAD, and we then construct a causal channel from a FIFO channel; the second step turns out to be considerably more involved. Our formalism for broadcast channels allows us to investigate the security of the traditional design paradigm that, as for TLS and SSH, builds a bidirectional channels from two unidirectional ones.

Forward-Secure Sequential Key Generators

Cryptographic designs are usually built around the assumption that honest parties hold some secret key that the adversary does not know, and any given guarantee of security vanishes instantaneously when this assumption is not fulfilled any more, i.e., an adversary manages to obtain a copy of the key material. This is not the case if the system is designed to offer forward security. In a nutshell, forward security means that the use of a cryptosystem is not affected by the potential exposure of the cryptosystem's key at some point in the future. For example, a symmetric encryption scheme is forward-secure if ciphertexts produced before the adversary gets the encryption key remain confidential. Forward security is considered a fundamental property for authenticated key exchange (AKE) protocols: here, it requires that session keys derived prior to corruption of participants are still safe to use. Perhaps surprisingly, cryptographic models for secure channels (e.g., [BKN02] and its follow-ups) do not incorporate forward security.

Nevertheless, the importance of forward-secure communication seems to be highly recognized by practitioners. For instance, the TLS 1.3 specification [Res16] hints to forward security as one of the targeted properties of the record protocol, and messaging applications like the Off-the-Record (OTR) [OTR16] and the more recent TexSecure [Tex14] protocols explicitly list it as one of their goals.

It is considered folklore that a forward-secure symmetric-key primitive can be obtained from the corresponding 'regular' primitive by frequently replacing the secret key with a fresh one. Bellare and Yee (BY) [BY03] show how to bootstrap from a regular pseudorandom generator (PRG) a so-called stateful generator, a deterministic algorithm that takes an initial random seed and expands it into a sequence of random-looking keys. They also build forward-secure symmetric encryption and message authentication codes (MACs) from the latter primitive.

In Chapter 7 we define sequential key generators (SKGs) as an alternative to stateful PRGs. We propose in particular a new security model for SKGs that, compared to the model of [BY03], is more appropriate for concrete applications where multiple participants run the SKG on the same initial seed (and thus generate the same key sequence). Note that this level of synchrony is required when remote parties use such generators, as symmetric primitives need keys to match on both sides. Our model lets the adversary see the generated keys in an adaptive fashion, independently of the order in which they are generated, while BY's model limits the adversary to obtain keys incrementally, thus implicitly modeling that participants are always perfectly synchronized with their key updates. Surprisingly, the higher degree of freedom that the attacker has does not result in a strictly stronger notion: as we prove in this thesis, our model is in fact equivalent to that of [BY03].

In Chapter 8 we define an enhanced version of SKGs, called seekable sequential key generators (SSKGs), that additionally provide random access to the sequence of generated keys. We propose two generic constructions of SSKGs and prove their security. The first construction is based on a novel primitive, a *shortcut one-way permutation* (ScP), and enjoys forward security in the random oracle model down to the one-wayness of the ScP. We also propose two concrete instantiations of ScPs, based on the hardness of RSA and of factoring. The second construction only relies on a regular PRG as building block, and is considerably more efficient than the first.

The property of seekability is particularly advantageous in the setting of secured local logging, where an SKG is used in combination with a MAC scheme to authenticate log files. Protecting the authenticity of log files, indeed, was the main motivation for the need of seekability, as described in [MP13, MP14]. In the context of computer forensics, in a local logging scenario, log entries are stored on each of the machines to be monitored, together with an authentication tag for each entry to ensure the integrity of the latter. To prevent an attacker that manages to break into the system from manipulating log entries created prior to the intrusion, the keys used to produce the authentication tags are frequently updated using an SKG. The property of seekability allows a system administrator to verify the authenticity of a specific subset of entries without the need to recover all authentication keys iteratively.

Further Related Work

Authenticated Encryption and Secure Channels. An approach towards cryptographic channels from the perspective of composability with other primitives is pursued in [CK01, CK02, PW01, MRT12]. For instance, Canetti and Krawczyk [CK02] consider channels in the UC framework. Prior work [CK01] by the same authors has a slightly more restricted model but receives a closer look by Namprempre [Nam02] who characterizes (game-based) notions that suffice to achieve a UC secure channel as per [CK01]. Pfitzmann and Waidner [PW01] consider aspects of composability in their paper on concurrent *secure message transmission*

(for stateless security notions), and Maurer *et al.* [MRT12] model cryptographic channels from the point of view of Constructive Cryptography. With the analysis of TLS in mind, Jager et al. [JKSS12] developed the ACCE security notion, which combined the security of key exchange and the subsequent use of the resulting session keys in a secure channel. Their work builds on that of Paterson et al. [PRS11], who introduced extensions of the standard AEAD notions to allow for length hiding; the ACCE approach was subsequently adopted and extended by Krawczyk et al. [KPW13] to analyze a wider set of TLS Handshake Protocol options. Again in the constructive cryptography setting, Badertscher et al. [BMM+15] propose an abstraction of a secure channel that should represent a high-level counterpart of an AEAD scheme, showing that (a modified version of) the TLS 1.3 Record Protocol fulfills the corresponding security goal.

Causality of events in distributed systems. Starting with Lamport's groundbreaking work on (distributed) logical clocks [Lam78], the role of causality in communication systems has been extensively investigated in the distributed computing community. For a survey on logical time, its connections to causality, and related notions we suggest the work by Schwarz and Mattern [SM94]. Cryptographic challenges related to causality have been recognized and studied extensively. Although targeting more general problems arising in the context of secure replication of services, Reiter and Birman [RB94] consider attacks on causality that specifically exploit causality violations among service requests, and propose countermeasures. In the same vein, Reiter and Gong [RG95] highlight the importance of detecting causal relations between messages exchanged by distributed processes; they introduce the notions of *causality denial*, i.e., making a server believe that a pair of causally related messages is not in such a relation, and *causality forgery*, i.e., convincing a server of a causality relation which does not exist. More recently Cachin, Kursawe, Petzold and Shoup [CKPS01] combined different techniques from modern cryptography and distributed computing to improve secure solutions of service replication. Among several variants of reliable broadcast primitives they propose a *causal broadcast* protocol which tolerates a Byzantine adversary and offers *input causality*—a property introduced in [RB94] ensuring that honest servers deliver client-requests in the right order and that exchanged messages remain secret until delivery.

We observe that while many of the works discussed above employ cryptographic techniques for solving problems from distributed computing, none of them addresses what we are interested in: the secure exchange of messages in multi-directional channels. We finally remark that functionality and security issues related to our concepts of causal channels and history integrity were recognized, but not formalized, by Goldberg, Ustaoglu, Van Gundy and Chen [GUVC09] in the context of designing multiparty chat protocols.

Chapter 2

Notation and Definitions

In this chapter we introduce the basic notation used in this thesis as well as background concepts and definitions, such as symmetric encryption, that may be helpful to understand the content of the next chapters.

2.1 General Notation

If X is a set we denote its cardinality by $|X|$. Let Σ be a finite set. We denote by Σ^* the set of strings containing symbols in Σ. In particular, $\{0,1\}^*$ denotes the set of binary strings. Given a string $s \in \Sigma^*$ we denote by $|s|$ its length, i.e., the number of symbols it contains. We denote the empty string by ε. Given two strings $s, t \in \Sigma^*$ we denote by $s \,\|\, t$ their concatenation. If $s \in \{0,1\}^*$ we denote by $\overline{s}$ the bit-inverse of s. For $n \in \mathbb{N}$ and a vector $\mathbf{v} = (v_1, \ldots, v_n)$ we say that n is the length of $\mathbf{v}$ and denote it by $|\mathbf{v}|$. If $n = 0$ we say that the vector is empty, and indicate this by writing $\mathbf{v} = ()$. For all $0 \le i \le j \le |\mathbf{v}|$ we write $\mathbf{v}[i, \ldots, j] = (v_i, \ldots, v_j)$. Slightly overloading notation, we denote by $\mathbf{v} \,\|\, \mathbf{w}$ the vector obtained by appending to $\mathbf{v}$ the components of $\mathbf{w}$, i.e., $(v_1, \ldots, v_n, w_1, \ldots, w_m)$. For $n, m \in \mathbb{N}$, $n \le m$, we denote by $[n\,..\,m]$ the set of integers $\{i : n \le i \le m\}$, and by $[\![n\,..\,m]\!]$ the set of pairs $\{(i,j) : i, j \in [n\,..\,m], i \ne j\}$. For $N \in \mathbb{N}$ we denote by $\mathbb{Z}_N^*$ the set of invertible elements of $\mathbb{Z}_N$, by $\varphi(N)$ its cardinality, and by $\mathbb{QR}_N$ the set of quadratic residues modulo N, i.e., $\mathbb{QR}_N = \{x^2 : x \in \mathbb{Z}_N^*\}$. We write $x \leftarrow y$ to assign value y to variable x. This is different from $x = y$ which indicates that variables x and y have the same value. We use the symbol $:=$ to define functions; for instance, $f(x) := x$ for all $x \in X$ indicates the identity function on domain X. If X is a finite set we write $x \leftarrow_\$ X$ for sampling x uniformly at random from X. We write $y \leftarrow_\$ \mathcal{A}(x)$ to indicate that a randomized algorithm $\mathcal{A}$ is run on input x and internal coin tosses and outputs y. If the algorithm is deterministic we emphasize this by writing $y \leftarrow \mathcal{A}(x)$. Within an algorithm specification we use the instruction 'Return y' to instruct the algorithm to halt with output y; we may also write simply 'Return' to indicate that no output will be produced. All logarithms are assumed to be to base 2 unless stated otherwise.

Defining security. We define security in the asymptotic setting and view runtime of algorithms and probabilities as functions of a security parameter $\lambda \in \mathbb{N}$ (which implicitly describes recommended key-lengths) that is conventionally passed to algorithms in unary form. We say that an algorithm is *efficient* if it runs in (probabilistic) polynomial time, i.e., if its runtime is in $\mathcal{O}(\lambda^c)$ for some $c \in \mathbb{N}$. We say that a function $f\colon \mathbb{N} \to \mathbb{R}$ is *negligible* if it is bounded by the inverse of any polynomial, i.e., if $|f(\lambda)| \in \lambda^{-\omega(1)}$. At a high level, we say that a cryptosystem is secure if its security can be efficiently violated only with negligible probability.

In this thesis we define security following the game-based tradition. A security game is a probabilistic experiment in which an adversary $\mathcal{A}$ plays against a challenger. The game shall encode what it means for a given cryptographic scheme to be secure. To this end, the challenger runs $\mathcal{A}$ as one of its subroutines and executes specific instructions that let $\mathcal{A}$ interact with the cryptographic scheme under consideration; eventually it outputs a bit $b \in \{0,1\}$ as outcome of the game, which determines whether the adversary won the game or not. If the adversary wins the game, this means that it successfully violated the security of the scheme. Suppose a security experiment consists of instructions $I_1, I_2, \ldots, I_n$ (in this order); we write $\Pr[b = 1 : I_1; I_2; \ldots; I_n]$ for the probability that an execution of this experiment terminates with outcome 1, where the probability is taken over the random choices of the experiment and the adversary's randomness; for a more compact notation we may also write $\Pr[\mathrm{E} = 1]$ where E is a label for the above experiment. Within a specific security experiment we use the instruction 'Terminate with b' to indicate that the challenger terminates with outcome b. We may also use the shortcut 'Require C' for the instruction that forces termination of the experiment with output 0 in case the condition C is not met, i.e., 'If $\neg C$: Terminate with 0'. When a security experiment terminates prematurely because the adversary triggers some particular event, we may write that the experiment 'penalizes' the adversary (as we will see, premature termination always implies that the adversary loses).

2.2 Cryptographic Assumptions and Primitives

2.2.1 Number-Theoretic Assumptions

The SQRT Assumption. Let N be a Blum integer, i.e., $N = pq$ for primes p, q such that $p \equiv q \equiv 3 \bmod 4$. It is well-known [MvV97] that the squaring operation $x \mapsto x^2 \bmod N$ is a permutation on $\mathbb{QR}_N$. Moreover, computing square roots in $\mathbb{QR}_N$, i.e., inverting this permutation, is as hard as factoring N. This intuition is the basis of the following hardness assumption.

Definition 1 (SQRT assumption)**.** *For any probabilistic algorithm* $\mathsf{SQRTGen}$ *that takes as input security parameter* 1^λ *and outputs tuples* (N, p, q, φ) *such that* $N = pq$*, factors* p *and* q *are prime and satisfy* $p \equiv q \equiv 3 \bmod 4$*, and* $\varphi = \varphi(N)$*, the* SQRT problem *is said to be* hard *if for all efficient adversaries* $\mathcal{A}$ *the success probability*

$$\Pr\left[x^2 = y \bmod N \;:\; (N, p, q, \varphi) \leftarrow_\$ \mathsf{SQRTGen}(1^\lambda);\; y \leftarrow_\$ \mathbb{QR}_N;\; x \leftarrow_\$ \mathcal{A}(1^\lambda, N, y)\right]$$

is negligible in λ*, where the probability is taken over the random choices of the experiment (including* $\mathcal{A}$*'s randomness). The* SQRT assumption *states that there exists an efficient probabilistic algorithm* $\mathsf{SQRTGen}$ *for which the SQRT problem is hard.*

The RSA Assumption. Let $N = pq$ for primes p, q, let $\varphi = \varphi(N)$, and let $e > 0$ be an integer such that $\gcd(e, \varphi) = 1$. The exponentiation to the e-th power modulo N, i.e., $x \mapsto x^e \bmod N$, is a permutation on $\mathbb{Z}_N^*$ [MvV97], and it is widely believed that inverting this permutation is hard.

Definition 2 (RSA assumption)**.** *For any probabilistic algorithm* RSAGen *that takes as input security parameter* 1^λ *and outputs tuples* (N, φ, e) *such that* $\varphi = \varphi(N)$ *and* $\gcd(e, \varphi) = 1$*, the* RSA problem *is said to be* hard *if for all efficient adversaries* $\mathcal{A}$ *the success probability*

$$\Pr\left[x^e = y \bmod N \;:\; (N, \varphi, e) \leftarrow_\$ \mathsf{RSAGen}(1^\lambda);\; y \leftarrow_\$ \mathbb{Z}_N^*;\; x \leftarrow_\$ \mathcal{A}(1^\lambda, N, e, y)\right]$$

is negligible in λ, where the probability is taken over the random choices of the experiment (including $\mathcal{A}$'s randomness). The RSA assumption *states that there exists an efficient probabilistic algorithm* RSAGen *for which the RSA problem is hard.*

2.2.2 One-Way Functions and Permutations

Intuitively, a one-way function (OWF) is easy to compute but hard to invert on average. More formally, f is one way if, given x chosen uniformly at random from the function's domain, computing $f(x)$ can be done efficiently while, given $f(x)$, finding a preimage cannot, but with negligible probability. We formalize this through an 'inverting' game where an adversary $\mathcal{A}$ has to find preimages of the function, as described below.

Definition 3 (One-Way Function)**.** *An efficiently computable function $f\colon \{0,1\}^* \to \{0,1\}^*$ is* one way *if the following advantage function is negligible for all efficient $\mathcal{A}$:*

$$\mathbf{Adv}^{\mathrm{OWF}}_{f,\mathcal{A}}(\lambda) := \Pr\left[\mathcal{A}(1^\lambda, f(x)) \in f^{-1}(f(x)) \ : \ x \leftarrow_\$ \{0,1\}^\lambda\right] \ ,$$

where the probability is taken over the choice of x as well as over $\mathcal{A}$'s randomness.

A one-way permutation (OWP) is a one-way function $\pi\colon \{0,1\}^* \to \{0,1\}^*$ that is length-preserving, i.e., $|\pi(x)| = |x|$ for all $x \in \{0,1\}^*$, and one-to-one. For OWPs it follows directly from the definition that every $y \in \{0,1\}^*$ uniquely determines its preimage $x = \pi^{-1}(y)$.

Concretely, one-way functions and permutations are built from (assumed) hard problems, e.g., the RSA problem (see Section 2.2). In fact, from RSA we can construct a family of one-way permutations. At a high level, a family of OWP consists of a tuple of efficient algorithms $(\mathsf{Gen}, \mathsf{Samp}, \mathsf{Eval})$ as follows. The parameter generation algorithm creates some public parameters $P \leftarrow_\$ \mathsf{Gen}(1^\lambda)$; each OWP defines a set D_P and a permutation $\pi_P\colon D_P \to D_P$. The sampling algorithm Samp takes as input P and selects uniformly at random an element $x \in D_P$ and outputs such element. Finally, the evaluation algorithm Eval effectively implements the permutation π_P: on input parameters P and an element $x \in D_P$, this deterministic algorithm returns $\pi_P(x)$. Security of the family requires each π_P to be one way. Note that a OWP family can be readily derived from the RSA assumption by setting $P = (N, e)$, $D_P = \mathbb{Z}^*_N$, and $\pi_P(x) := x^e \bmod N$. A similar observation holds for the SQRT assumption.

2.2.3 Pseudorandom Generators

A *pseudorandom generator* (PRG) is an efficiently computable function that stretches a truly random string into a longer, 'random-looking' string. In other words, a PRG deterministically expands a small amount of true randomness into a larger amount of pseudorandomness. Formally, saying that a string 'looks random' means that it is computationally indistinguishable from a uniformly chosen string of the same length.

Definition 4 (Pseudorandom Generator)**.** *Let $\ell\colon \mathbb{N} \to \mathbb{N}$ be a polynomial and let $G\colon \{0,1\}^\lambda \to \{0,1\}^{\ell(\lambda)}$ be an efficiently computable function. We say that G is a* pseudorandom generator *if for all $\lambda \in \mathbb{N}$ we have $\ell(\lambda) > \lambda$ and for all efficient adversaries $\mathcal{A}$ the following advantage function is negligible:*

$$\mathbf{Adv}^{\mathrm{PRG}}_{G,\mathcal{A}}(\lambda) = \left|\Pr\left[\mathcal{A}(1^\lambda, G(s)) = 1 \ : \ s \leftarrow_\$ \{0,1\}^\lambda\right] - \Pr\left[\mathcal{A}(1^\lambda, y) = 1 \ : \ y \leftarrow_\$ \{0,1\}^{\ell(\lambda)}\right]\right| \ ,$$

where the probabilities are taken over the random choices of s, y, and over $\mathcal{A}$'s randomness. The string s is usually called seed *and ℓ is called the* expansion *of the pseudorandom generator.*

2.2.4 Random Oracles

It is often difficult to design cryptographic schemes that are efficient enough for deployment and also provably secure under well-studied assumptions. In these cases, when a proof of security seems to be out of reach, it may be helpful to consider an idealized model in which a security reduction can be found. One of such idealizations is the random oracle model, which assumes the existence of a truly random function $H\colon \{0,1\}^* \to \{0,1\}^*$ that can be publicly evaluated by querying an oracle. That is, everybody (honest parties as well as adversaries) can pose queries of the form $x \in \{0,1\}^*$ to the random oracle and will instantaneously obtain as response the corresponding value $H(x)$. Using a random oracle H eases the security proof for two reasons: as long as the input x is not new, the output $H(x)$ can be assumed to be random; moreover, H can be efficiently simulated by a reduction via lazy sampling, i.e., by picking a fresh random value $y \in \{0,1\}^n$ for any new query $x \in \{0,1\}^n$ and setting $H(x) \leftarrow y$, or by returning the previously chosen image $y = H(x)$ in case x was already queried to the oracle. Clearly, in practice there exists no such a function like a random oracle. However, many researchers believe that having a proof in the random oracle model is better than nothing. The so-called random oracle methodology, pioneered by Bellare and Rogaway [BR93], consists in proving the security of a cryptographic scheme in the random oracle model and then 'instantiating' the random oracle with a concrete cryptographic hash function.

2.3 Cryptographic Models for Secure Channels

2.3.1 Symmetric Encryption

Consider two parties that share some secret key and wish to communicate. A *symmetric encryption* scheme (SE) allows the sender to use the shared secret to turn messages into ciphertexts and the receiver to recover from those ciphertexts the original messages. Moreover, it is hard for anyone who does not know the key to infer from ciphertexts the underlying messages, or any information about them.

Formally, a symmetric encryption scheme is specified by a key space $\mathcal{K}$, a message space $\mathcal{M} \subseteq \{0,1\}^*$, a ciphertext space $\mathcal{C}$, and algorithms KeyGen, Enc, and Dec as follows. The key generation algorithm KeyGen receives as input a security parameter and generates a symmetric key $K \in \mathcal{K}$; the encryption algorithm Enc takes a key K and a message $m \in \mathcal{M}$ as input, and outputs a ciphertext $c \in \mathcal{C}$; the decryption algorithm Dec takes as input a key K and a ciphertext $c \in \mathcal{C}$, and outputs a message $m \in \mathcal{M}$ or, optionally, a distinguished symbol $\perp \notin \mathcal{M}$ to indicate that an error occurred. If Dec returns a message $m \in \mathcal{M}$ we say that the algorithm accepts, otherwise we say that it rejects.

The encryption and decryption algorithms may additionally use random coins and process auxiliary input such as a nonce ('number used once') or an initialization vector (IV). Either of the above choices leads to different SE variants: *randomized*, *nonce-based*, and *IV-based* symmetric encryption. Without any further mention, in this thesis we assume that all encryption algorithms are randomized and the decryption algorithms are deterministic. Correspondingly, for $K \leftarrow_\$ \mathsf{KeyGen}(1^\lambda)$, $m \in \mathcal{M}$, $m' \in \mathcal{M} \cup \{\perp\}$, and $c, c' \in \mathcal{C}$, we write $c \leftarrow_\$ \mathsf{Enc}_K(m)$ and $m' \leftarrow \mathsf{Dec}_K(c')$. We also require all schemes to fulfill the following correctness condition: for every key $K \leftarrow_\$ \mathsf{KeyGen}(1^\lambda)$, every message $m \in \mathcal{M}$ and ciphertext $c \leftarrow_\$ \mathsf{Enc}_K(m)$ it holds $\mathsf{Dec}_K(c) = m$.

The security goal of symmetric encryption, called *confidentiality*, is modeled in the game-based tradition via indistinguishability experiments. Such games consider a powerful adversary that can obtain encryptions of messages of its choosing by querying an encryption oracle, in which case we talk of a *chosen-plaintext attack*, or in addition can obtain decryptions of ci-

phertexts of its choosing by interacting with a decryption oracle, in a *chosen-ciphertext attack*. The encryption oracle is a so-called 'left-or-right' oracle, which can be queried with pairs of messages (m_0, m_1), and answers each of these queries by always returning either an encryption of the 'left' message m_0 or an encryption of the 'right' message m_1, depending on a secret bit b; the goal of the adversary is to guess b. Later in this section we will come back to SE security and provide a detailed description of its security games.

2.3.2 Authenticated Encryption (with Associated Data)

While being often considered the most fundamental security goal, confidentiality alone is not all what is needed for secure communication. In many practical scenarios, beyond the assurance that only the intended recipient may read encrypted messages, it is also essential for the recipient to know that a given message really originated from the alleged sender and that it was not adversarially modified. With other words, *integrity* of messages is a natural target for achieving secure communication. A symmetric encryption scheme that offers both confidentiality and integrity is called an *authenticated encryption* (AE) scheme. The integrity property essentially says that Dec should only return messages that were encrypted by the sender. This property, called *integrity of plaintexts*, is modeled via a security experiment that lets $\mathcal{A}$ obtain encryptions of messages of its choice as well as decryptions of ciphertexts of its choice, again using corresponding oracles; the goal of the adversary is to make the decryption oracle output a message that has not been previously queried to the encryption oracle. A stronger flavor of integrity, called *integrity of ciphertexts*, declares the adversary successful if it produces a ciphertext that is accepted and that was not output by the encryption oracle.

As emerged from real-world applications, sometimes it is necessary to transmit along with a ciphertext some additional information that needs not be kept confidential but should be protected against modification (e.g., routing information should be authentic, but also needs to be visible to all intermediate nodes), or to bind the ciphertext to a given context. To this end, an AE scheme *with associated data* (AEAD) may be employed. Syntactically, an AEAD scheme is similar to a symmetric encryption scheme, the only difference being that Enc and Dec take an additional input ad from some associated data space $\mathcal{AD}$. Originally AEAD was introduced in [Rog02] as a variant of nonce-based encryption, however, its syntax can be flexibly extended to randomized and stateful algorithms. In accordance with the notions of symmetric encryption previously introduced, in this thesis we consider randomized and stateful AEAD, and correspondingly write $c \leftarrow_{\$} \mathsf{Enc}_K(ad, m)$ and $m \leftarrow \mathsf{Dec}_K(ad, c)$. No matter how it is transmitted, the associated data is supposed to match on both sides. Correctness for AEAD demands that for every key $K \leftarrow_{\$} \mathsf{KeyGen}(1^\lambda)$, every message $m \in \mathcal{M}$, every associated data $ad \in \mathcal{AD}$, and every ciphertext $c \leftarrow_{\$} \mathsf{Enc}_K(ad, m)$ we have that $\mathsf{Dec}_K(ad, c) = m$. AEAD security is similar in spirit to AE security, and is thus a combination of confidentiality and integrity. The corresponding security games are a refinement of the indistinguishability and integrity experiments mentioned above that let $\mathcal{A}$ control the associated data used in encryption and decryption queries.

2.3.3 Stateful Symmetric Encryption

Authenticated Encryption (with Associated Data) on its own does not constitute a secure channel. Beyond offering confidentiality and integrity, a secure channel should also ensure that messages are not replayed or reordered. Bellare, Kohno, and Namprempre [BKN02] introduced the notion of stateful encryption to explicitly target these goals.

A *stateful encryption* scheme is a symmetric encryption scheme that additionally protects against replay and reordering attacks. This is usually done by letting the sender and the receiver

keep some state information, e.g., a counter, that is synchronized on both sides at the moment of initialization, and updated upon encryption and decryption invocations. Formally, let us define a stateful encryption scheme from an SE scheme by introducing a state space $\mathcal{S}$ and replacing algorithm KeyGen with an initialization algorithm Init that generates not only the key but also encryption and decryption states. Further, let Enc and Dec take the encryption state (or sending state), respectively, the decryption state (or receiving state), as additional inputs, and output updated states. We write $(K, st_S, st_R) \leftarrow_{\$} \mathsf{Init}(1^\lambda)$, $(st'_S, c) \leftarrow_{\$} \mathsf{Enc}_K(st_S, m)$ and $(st'_R, m) \leftarrow \mathsf{Dec}_K(st_R, c)$, respectively. Correctness for stateful encryption requires that if Enc and Dec have synchronized states (e.g., after initialization), Enc is invoked sequentially on input some messages $m_1, m_2, \ldots$, and the resulting ciphertexts $c_1, c_2, \ldots$ are then processed by Dec in the same order they were produced, then the original message sequence $m_1, m_2, \ldots$ is recovered. Note that the correctness condition for stateful SE is milder than that for SE (which promises message recovery regardless of the order in which decryptions are performed).

The syntax of stateful SE can be augmented to support associated data in the natural way. We stress that when referring to a *stateful SE* scheme we mean that both Enc and Dec keep state.[1] If not clear from the context, we highlight that a symmetric encryption scheme is *not* stateful (i.e., it does not fulfill the syntax just defined) by naming it a *stateless* SE scheme. We will see next that security notions for stateful encryption are similar to, but more complex than, the corresponding notions for stateless encryption.

2.3.4 Security Notions for Symmetric and Authenticated Encryption

In this section we formalize the security of SE, AE, AEAD and their stateful variants. For all primitives we define confidentiality via indistinguishability games, namely *indistinguishability under chosen-plaintext attacks* (IND-CPA) for modeling passive adversaries, and *under chosen-ciphertext attacks* (IND-CCA) for modeling active adversaries. We also define two flavors of integrity, called *plaintext integrity* (INT-PTXT) and *ciphertext integrity* (INT-CTXT), respectively. All notions are in the style of [BDJR97].

Stateless notions. In the IND-CPA game the adversary has access to a left-or-right encryption oracle $\mathcal{O}_{\mathsf{LoR}}$ that it can query multiple times. A left-or-right query consists of a pair of messages of the same length, m_0 and m_1, and (if applicable) associated data ad; the oracle answers by returning an encryption of m_b (obtained using ad as associated data), where $b \in \{0, 1\}$ is not known to $\mathcal{A}$.

The IND-CCA game also provides a decryption oracle $\mathcal{O}^*_{\mathsf{Dec}}$ that the adversary can query on any arbitrary pair (ad, c) of associated data and ciphertext, and obtain the corresponding decryption, with one exception: in case any prior left-or-right query (m_0, m_1) with associated data ad produced ciphertext c, the oracle returns as a response the special symbol '$\diamond$' instead, meaning that the output of the decryption algorithm has been artificially suppressed. This is to avoid trivial attacks: in the above setting decryption by correctness would lead to $\mathsf{Dec}_K(ad, c) = m_b$, which tells $\mathcal{A}$ the value of b right away.

The integrity experiments let the adversary interact with the encryption and decryption routines through oracles $\mathcal{O}_{\mathsf{Enc}}$ and $\mathcal{O}_{\mathsf{Dec}}$ which, upon request, return encryptions $c \leftarrow_{\$} \mathsf{Enc}_K(ad, m)$ and decryptions $m \leftarrow \mathsf{Dec}_K(ad, c)$ for arbitrary chosen (ad, m) and (ad, c). In the INT-PTXT game we declare the adversary successful if it submits to $\mathcal{O}_{\mathsf{Dec}}$ a pair (ad^*, c^*) that decrypts to a valid message m^* (i.e., $m^* \neq \perp$) such that no query (ad^*, m^*) was previously posed to $\mathcal{O}_{\mathsf{Enc}}$.

[1] Some schemes have a stateful encryption algorithm but a stateless decryption algorithm: according to our convention, these fall in the class of 'standard' (a.k.a. *stateless*) SE schemes.

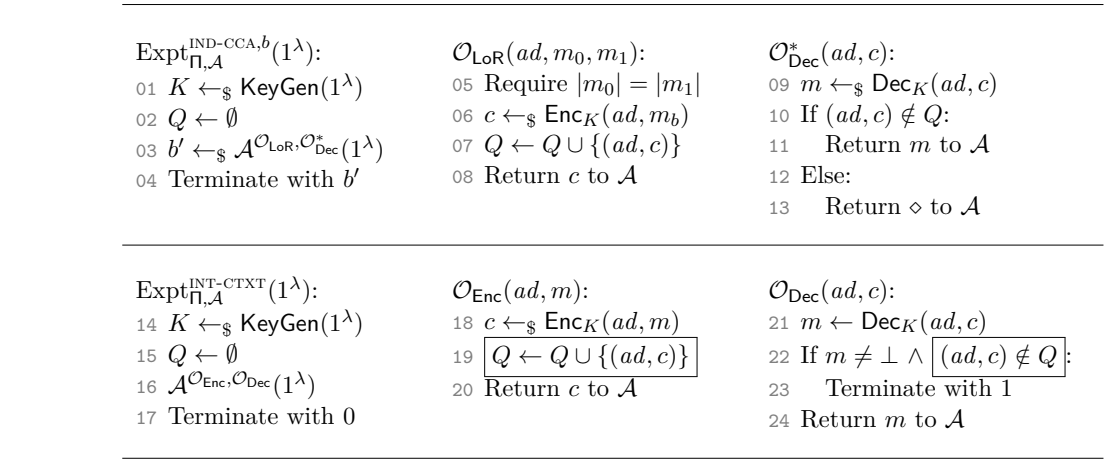


Figure 2.1: *AEAD security experiments for confidentiality and integrity. The set Q is maintained by the experiment for bookkeeping pairs (ad, c) of associated data and ciphertext generated within each encryption call. We assume $ad \in \mathcal{AD}$, $m_0, m_1, m \in \mathcal{M}$, and $c \in \mathcal{C}$ for all such values provided by the adversary. The IND-CPA experiment can be readily obtained from the IND-CCA game by omitting the decryption oracle $\mathcal{O}^*_{\mathsf{Dec}}$. Similarly, the INT-PTXT experiment can be derived from the INT-CTXT experiment by replacing the boxed text of lines 19 and 22 with instructions '$Q \leftarrow Q \cup \{(ad, m)\}$' and '$(ad, m) \notin Q$'. Notice that by ignoring the associated data ad we obtain confidentiality and integrity games for authenticated encryption.*

Similarly, the adversary wins the INT-CTXT game if it submits a valid pair (ad^*, c^*) for which no previous encryption query (ad^*, m^*) resulted in ciphertext c^*.

In Figure 2.1 we give full details of the indistinguishability experiments (upper row) and integrity experiments (lower row). Given the experiments, we formalize security in Definition 5.

Stateful notions. Security for stateful encryption may be defined starting from stateless SE security by modifying the security experiments such that they also capture protection against replay and reordering attacks. Beyond the obvious syntactical modification that replaces the KeyGen with Init and turns Enc and Dec into stateful algorithms, the IND-CPA game remains essentially unchanged. Indeed, replay and reordering attacks concern active adversaries, and hence cannot be described without a decryption oracle. All the other security experiments, IND-CCA, INT-PTXT, and INT-CTXT, need a major adaptation to the stateful setting.

We start with describing the IND-CCA experiment. Recall that in the stateless setting (Figure 2.1) the adversary can query the decryption oracle on any pair (ad, c) of associated data and ciphertext but, in case the ciphertext c was previously returned by the left-or-right oracle in response to a query with associated data ad, the oracle $\mathcal{O}^*_{\mathsf{Dec}}$ gives the adversary the suppression symbol '$\diamond$' to prevent trivial wins. In the stateful setting this suppression mechanism needs to be adapted to the functionality of a stateful SE scheme. To this end, as long as $\mathcal{A}$ submits for decryption a prefix of the *sequence* of ciphertexts (and associated data) produced by the left-or-right oracle, the oracle $\mathcal{O}^*_{\mathsf{Dec}}$ suppresses the output of Dec as it would otherwise reveal m_b, and hence b, by correctness. When $\mathcal{A}$ queries $\mathcal{O}^*_{\mathsf{Dec}}$ about the first pair (ad, c) that deviates from the genuine sequence produced by $\mathcal{O}_{\mathsf{LoR}}$, the algorithms Enc and Dec lose synchronization and correct decryption is no longer guaranteed: from this moment on the attack becomes *active* and the adversary is given the actual output of Dec (in particular, $\mathcal{O}^*_{\mathsf{Dec}}$ stops suppressing messages). Observe that, as the correctness requirement for stateful encryption is *milder* than for stateless encryption (genuine ciphertexts decrypt to the correct messages only if they are fed to Dec in the same order as they were produced by Enc), the stateful decryption

oracle suppresses *less* than the stateless one.

Based on the above observation it is now easy to define stateful integrity. The adversary wins in the INT-PTXT game if it causes the message sequence output by Dec to deviate from the message sequence input to Enc. Similarly, the adversary is successful in the INT-CTXT game if it submits for decryption an out-of-sync pair (ad, c) that is accepted by Dec.

In Figure 2.2 we specify the security experiments of indistinguishability (upper row) and integrity (lower row) for stateful encryption. Security is formalized in Definition 5.

$\mathrm{Expt}^{\text{IND-CCA},b}_{\Pi,\mathcal{A}}(1^\lambda)$:
01 $(K, st_S, st_R) \leftarrow_\$ \mathsf{Init}(1^\lambda)$
02 $s \leftarrow r \leftarrow 0$
03 $\mathsf{active} \leftarrow 0$
04 $b' \leftarrow_\$ \mathcal{A}^{\mathcal{O}_{\mathsf{LoR}}, \mathcal{O}^*_{\mathsf{Dec}}}(1^\lambda)$
05 Terminate with b'

$\mathcal{O}_{\mathsf{LoR}}(ad, m_0, m_1)$:
06 Require $|m_0| = |m_1|$
07 $s \leftarrow s + 1$
08 $(st_S, c) \leftarrow_\$ \mathsf{Enc}_K(st_S, ad, m_b)$
09 $ad_s \leftarrow ad$, $c_s \leftarrow c$
10 Return c to $\mathcal{A}$

$\mathcal{O}^*_{\mathsf{Dec}}(ad, c)$:
11 $r \leftarrow r + 1$
12 If $r > s \vee (ad, c) \neq (ad_r, c_r)$:
13 $\quad \mathsf{active} \leftarrow 1$
14 $(st_R, m) \leftarrow_\$ \mathsf{Dec}_K(st_R, ad, c)$
15 If $\mathsf{active} = 1$:
16 $\quad$ Return m to $\mathcal{A}$
17 Else:
18 $\quad$ Return $\diamond$ to $\mathcal{A}$

$\mathrm{Expt}^{\text{INT-CTXT}}_{\Pi,\mathcal{A}}(1^\lambda)$:
19 $(K, st_S, st_R) \leftarrow_\$ \mathsf{Init}(1^\lambda)$
20 $s \leftarrow r \leftarrow 0$
21 $\mathcal{A}^{\mathcal{O}_{\mathsf{Enc}}, \mathcal{O}_{\mathsf{Dec}}}(1^\lambda)$
22 Terminate with 0

$\mathcal{O}_{\mathsf{Enc}}(ad, m)$:
23 $(st_S, c) \leftarrow_\$ \mathsf{Enc}_K(st_S, ad, m)$
24 $s \leftarrow s + 1$
25 $ad_s \leftarrow ad$, $\boxed{c_s \leftarrow c}$
26 Return c to $\mathcal{A}$

$\mathcal{O}_{\mathsf{Dec}}(ad, c)$:
27 $(st_R, m) \leftarrow \mathsf{Dec}_K(st_R, ad, c)$
28 $r \leftarrow r + 1$
29 If $m = \perp$: Return $\perp$ to $\mathcal{A}$
30 Else if $r > s \vee \boxed{(ad, c) \neq (ad_r, c_r)}$:
31 $\quad$ Terminate with 1
32 Return m to $\mathcal{A}$

Figure 2.2: *Security experiments for stateful authenticated encryption. We assume $ad \in \mathcal{AD}$, $m_0, m_1, m \in \mathcal{M}$, and $c \in \mathcal{C}$ for all such values provided by the adversary. The integers s and r denote the number of encryption calls (sent messages) and of decryption calls (received messages) performed by the adversary. The IND-CPA experiment can be readily obtained from the IND-CCA game by omitting the decryption oracle $\mathcal{O}^*_{\mathsf{Dec}}$. Similarly, the INT-PTXT experiment can be derived from the INT-CTXT experiment by replacing the boxed text of lines 25 and 30 with instructions '$m_s \leftarrow m$' and '$(ad, m) \neq (ad_r, m_r)$'. Notice that by ignoring the associated data ad we obtain confidentiality and integrity games for stateful authenticated encryption (named INDCPA, IND-sfCCA, INT-sfPTXT and INT-sfCTXT in [BKN02]).*

Definition 5 (Stateless and stateful security for symmetric encryption)**.** *Let Π be an encryption scheme with associated data and let $\mathcal{K}$, $\mathcal{S}$, $\mathcal{M}$, $\mathcal{C}$, and $\mathcal{AD}$ be its key space, state space, message space, ciphertext space and associated data space, respectively. In case Π is stateless ignore the state space $\mathcal{S}$ and consider the experiments from Figure 2.1, otherwise consider those from Figure 2.2. For* $\text{ATK} \in \{\text{CPA}, \text{CCA}\}$ *we say that Π offers* ATK-indistinguishability *if for all efficient adversaries $\mathcal{A}$ the following advantage function is negligible,*

$$\mathbf{Adv}^{\text{IND-ATK}}_{\Pi,\mathcal{A}}(\lambda) = \left| \Pr\left[\mathrm{Expt}^{\text{IND-ATK},1}_{\Pi,\mathcal{A}}(1^\lambda) = 1\right] - \Pr\left[\mathrm{Expt}^{\text{IND-ATK},0}_{\Pi,\mathcal{A}}(1^\lambda) = 1\right] \right| \ .$$

Similarly, for $\text{ATK} \in \{\text{PTXT}, \text{CTXT}\}$ *we say that Π offers* ATK-integrity *if for all efficient adversaries $\mathcal{A}$ the following advantage function is negligible,*

$$\mathbf{Adv}^{\text{INT-ATK}}_{\Pi,\mathcal{A}}(\lambda) = \left| \Pr\left[\mathrm{Expt}^{\text{INT-ATK}}_{\Pi,\mathcal{A}}(1^\lambda) = 1\right] \right| \ .$$

*We abbreviate indistinguishability under chosen-plaintext attacks (*CPA*-indistinguishability) and chosen-ciphertext attacks (*CCA*-indistinguishability) by writing IND-CPA and IND-CCA, respectively. Similarly, for plaintext integrity (*PTXT*-integrity) and ciphertext integrity (*CTXT*-integrity) we use the shortcuts INT-PTXT and INT-CTXT.*

For consistency with the notation used in this thesis we abbreviate the security notions for stateless encryption as well as for stateful encryption using the same labels. We point out that the stateful notions are equivalent to the INDCPA, INDsfCCA, INTsfPTXT, and INTsfCTXT notions from [BKN02].

The relations among the above security notions for symmetric encryption are well understood, both in the stateless setting [BN00] and in the stateful setting [BKN02]. Confidentiality against active adversaries implies confidentiality against passive adversaries; shortly, IND-CCA $\Longrightarrow$ IND-CPA, and ciphertext integrity implies plaintext integrity, INT-CTXT $\Longrightarrow$ INT-PTXT. Moreover, confidentiality against passive adversaries, when combined with ciphertext integrity, implies confidentiality against active adversaries: IND-CPA $\wedge$ INT-CTXT $\Longrightarrow$ IND-CCA. For this reason, usually AEAD security is defined as the combination of IND-CPA and INT-CTXT.

2.3.5 Modeling Secure Channels

In this thesis we develop various notions of cryptographic channels: stream-based, FIFO, and causal channels. Our syntactical model assumes that channels support *send* and *receive* operations executed by the parties that use the channel, where the send operation transforms a message into a ciphertext and hands it over to the network layer, and the receive operation takes an incoming ciphertext and translates it back to a message. In our model, a 'message' is a data object that is seen and processed by an application, while a 'ciphertext' is a lower-level object that is transmitted over a network. If required, both the send and the receive operations can also take associated data that is assumed to match on both sides. The endpoints of a channel, i.e., the parties that communicate via the channel, are assumed to keep state between invocations of the algorithms.

Our syntax reflects the generic functionality that a channel should provide, i.e., allowing users to transmit messages and to obtain messages from other users in a reliable way. Note that, while a secure channel is generally realized from some authenticated encryption primitive, an authenticated channel might choose to leave confidentiality aside and provide only integrity. We prefer to keep a higher level of abstraction and explicitly separate the generic notion of a channel from its building blocks and hence replace encryption and decryption algorithms with sending (Send) and receiving (Recv) procedures.

Cryptographic channels necessarily require some key material for setup (for instance, a password, a pre-shared symmetric key, or a combination of public and secret keys that are used in interactive key agreement). As the technical details connected to channel initialization are irrelevant for our work, our syntax definition abstracts them away. We assume a centralized setup algorithm that generates the initial states for all participating users and distributes them securely. In practice, this function is likely implemented by some authenticated key exchange step.

All channels considered in the next chapters obey the syntax of Definition 6, adapted to the streaming setting and to the broadcast setting respectively.

Definition 6 (Generic syntax of channels)**.** *A* channel *with associated data space $\mathcal{AD}$, message space $\mathcal{M}$, ciphertext space $\mathcal{C}$, state space $\mathcal{S}$, and error space $\mathcal{E}$, with $\mathcal{M} \cap \mathcal{E} = \emptyset$, is a tuple* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *of efficient algorithms as follows:*

- Init. *This randomized algorithm takes as input a security parameter 1^λ and a number of participants $N \in \mathbb{N}$, and generates initial states $st_1, \ldots, st_N \in \mathcal{S}$, one per participant. We write $(st_1, \ldots, st_N) \leftarrow_\$ \mathsf{Init}(1^\lambda, N)$.*

- Send. *This algorithm takes as input a state $st \in \mathcal{S}$, associated data $ad \in \mathcal{AD}$, and a message $m \in \mathcal{M}$, and outputs a state $st' \in \mathcal{S}$, a ciphertext $c \in \mathcal{C}$, and optionally some*

auxiliary information aux. We write $(st', c, aux) \leftarrow_{\$} \mathsf{Send}(st, ad, m)$.

- Recv. *This algorithm takes as input a state* $st \in \mathcal{S}$*, an origin indicator* $j \in [1\,..\,N]$*, associated data* $ad \in \mathcal{AD}$*, and a ciphertext* $c \in \mathcal{C}$*, and outputs a state* $st' \in \mathcal{S}$*, an element in* $m \in \mathcal{M} \cup \mathcal{E}$*, and optionally some auxiliary information aux. We write* $(st', m, aux) \leftarrow_{\$} \mathsf{Recv}(st, j, ad, c)$*. In case* $m \in \mathcal{E}$ *we say that the algorithm rejects, otherwise that it accepts.*

For stream-based channels we assume a communication model in which a sender transmits data to a receiver; thus, all exchanged messages are sent in one direction, from the sender to the receiver. This can be expressed in our syntax by setting $N = 2$ and letting the initialization algorithm output a state for the sender, $st_1 = st_S$, and a state for the receiver, $st_2 = st_R$ (clearly, in this setting the sender only invokes algorithm Send and the receiver only invokes Recv). As we will see in Chapter 3, for stream-based channels algorithm Send also takes as additional input a flush flag $f \in \{0, 1\}$, which specifies if the sender should empty its buffer. For stream-based channels we also let the error space $\mathcal{E}$ be any non-empty set (as long as it is disjoint from the message space). This allows us to capture the fact that a channel may produce more than one error symbol within the model.

Broadcast channels instead are meant to be used by potentially more than two parties and, more importantly, each party can perform both send and receive operations. As it is not meaningful, in this setting, to refer to users as senders or receivers, in broadcast channels we instead refer to participants. We will define two variants of broadcast channels: FIFO channels (in Chapter 5) and causal channels (in Chapter 6). The algorithms Send and Recv of a causal channel return, besides a state and a message, respectively, a ciphertext, also a history as auxiliary output $aux = h$. Such history roughly describes the view that a user should have of the ongoing communication.

Chapter 3

Stream-Based Channels

In this chapter we develop functional specifications, security notions, and a construction for stream-based channels. The security models turn out to be quite complex. This seems to be a natural consequence of the increased degree of freedom of the stream-based channel algorithms (due to fragmentation and flushing) compared to traditional authenticated encryption primitives, that has to be reflected in the adversary's capabilities.

3.1 Introduction

Well-established cryptographic models like [BKN02] treat secure channels as providing an *atomic* interface for messages, meaning that the channel is designed only for sending and receiving sequences of messages, where each message is considered a unit. However, this only captures a fraction of secure channel designs that are actually used in the real world. In particular, TLS and SSH provide a *streaming* interface for the applications that use them: applications submit fragments of message streams to an API, and similarly receive fragments of message streams from the API. The sending algorithm may arbitrarily buffer and/or fragment its input, and so can the receiving algorithm. Thus, there is a mismatch between atomic descriptions of secure channels in the cryptography literature and the reality of how secure channels process their inputs. As one may expect, such mismatches can have negative consequences for security. A prominent example of this comes from the plaintext recovery attack against SSH given by Albrecht *et al.* [APW09]. Their attack specifically exploits the adversary's ability to deliver arbitrary sequences of SSH packet fragments to the receiver (over TCP/IP) and observe the receiver's behavior in response. The attack is possible despite the analysis of [BKN02] which proved that the SSH secure channel satisfies suitable *atomic* stateful security notions. Related attacks against certain IPsec configurations (and exploiting IPsec's need to handle IP fragmentation) were presented in [DP10]. To make the attack from [APW09] visible (within the security model) and prevent similar attacks, Boldyreva *et al.* [BDPS12] (BDPS) extended the classical, atomic encryption notions to cover the case of SSH-like stream-based secure channels, broadening the SSH-specific work of [PW10]. However, while BDPS allow for fragmented delivery of ciphertexts to the receiver, their work still assumes that the encryption process on the sender's side is atomic, meaning that there is a one-to-one correspondence between messages and ciphertexts produced by the sender. This may be the case for SSH when used in interactive sessions, but it is not necessarily the case for the tunneling mode of SSH and for other secure channels protocols.

In this Chapter we develop a new notion of channel that is stream-oriented. In contrast to previous cryptographic models for secure channels (with the exception of [BDPS12]) that

consider sending and receiving routines handling *atomic* messages, our syntax augments the interfaces of the algorithms Send and Recv to input and output fragments of a message stream. Our models extend those of [BKN02, BDPS12] to handle the streaming nature of the channels that we consider. While our methodology and modeling closely resembles that of [BDPS12], and indeed builds upon it, a crucial difference comes in our treatment of the sending (or encrypting) function of a stream-based channel: in [BDPS12], this is still atomic (while decryption is not), whereas in our stream-based channel setting both the sending and receiving function support streams of data, with potentially arbitrary buffering and fragmentation on the sending and receiving side. This requires careful modification, reconsideration, and rework of the confidentiality definitions of [BDPS12]. In addition, we develop suitable integrity notions for the streaming setting, whereas [BDPS12] does not consider this aspect. This is important because the (implicit) security properties that applications expect a secure channel to provide are confidentiality as well as integrity.

Before going on with the formal definition of a stream-based channel we need to set some notation.

String notation. Let Σ be an alphabet and let $s, t \in \Sigma^*$. Recall that $s \parallel t$ denotes the concatenation of strings s and t. We say that s is a *prefix* of t and write $s \preccurlyeq t$ if there exists $r \in \Sigma^*$ such that $s \parallel r = t$; in this case we write $r = t \,\%\, s$ and say that r is the remainder of t modulo s. We write $s \prec t$ to indicate that s is a strict prefix of t, i.e., $s \preccurlyeq t$ and $s \neq t$. We denote the longest common prefix of s and t by $[s, t] = [t, s]$. Note that $s \preccurlyeq t$ if and only if $[s, t] = s$. Throughout this chapter we will often encounter expressions of the form $s \,\%\, [s, t]$, that is, the suffix of s with the longest common prefix of s and t stripped off. For example, if $s = 01001$ and $t = 0101101$ then $[s, t] = [t, s] = 010$, $s \,\%\, [s, t] = 01$ and $t \,\%\, [s, t] = 1101$. We also anticipate that in what follows we may overload the notation just described and use it between strings that belong to different alphabets. In this case, the alphabets are assumed to be disjoint and the resulting string is over the union of the alphabets. For instance, we may write $m \parallel \perp$ for the concatenation of a bit-string $m \in \{0,1\}^*$ and an 'error symbol' $\perp \notin \{0,1\}^*$. If $\mathbf{v} = (v_1, \ldots, v_n)$ is a vector of strings, i.e., $\mathbf{v} \in (\Sigma^*)^*$, we write $\parallel \mathbf{v}$ for the concatenation $v_1 \parallel \ldots \parallel v_n$ mapping the entries of $\mathbf{v}$ to a single string, and conventionally define $\parallel () = \varepsilon$. For example, if $\mathbf{v} = (110, 0110, 01)$ then $\parallel \mathbf{v} = 110011001$.

3.2 Syntax and Functionality

As for traditional cryptographic channels induced by stateful AE schemes, a stream-based channel should allow secure transmission of data between two parties that share secret key material. In contrast to the former schemes, however, the data to be transmitted is a *stream* rather than a sequence of messages. From a functionality perspective, we view stream-based channels as a mathematical abstraction of reliable network protocols like the Transmission Control Protocol (TCP) [Pos81]. In what follows we formalize what 'transmitting a stream of data' actually means.

For stream-based channels we define message space and ciphertext space to be the set of bit strings, i.e., $\mathcal{M} = \mathcal{C} = \{0,1\}^*$, as it is usually the case for real-world channel implementations. In fact, we understand messages and ciphertexts not as atomic units but as fragments (i.e., substrings) of a message stream and of a ciphertext stream. Additionally, we do not stipulate a particular input/output behavior on the sender side but allow the sending algorithm Send to process input data at its discretion, e.g., implementing some form of buffering. This is different from the approach chosen by Boldyreva *et al.* [BDPS12] for modeling symmetric encryption supporting ciphertext fragmentation. There, fragmentation may occur at the receiver, while the

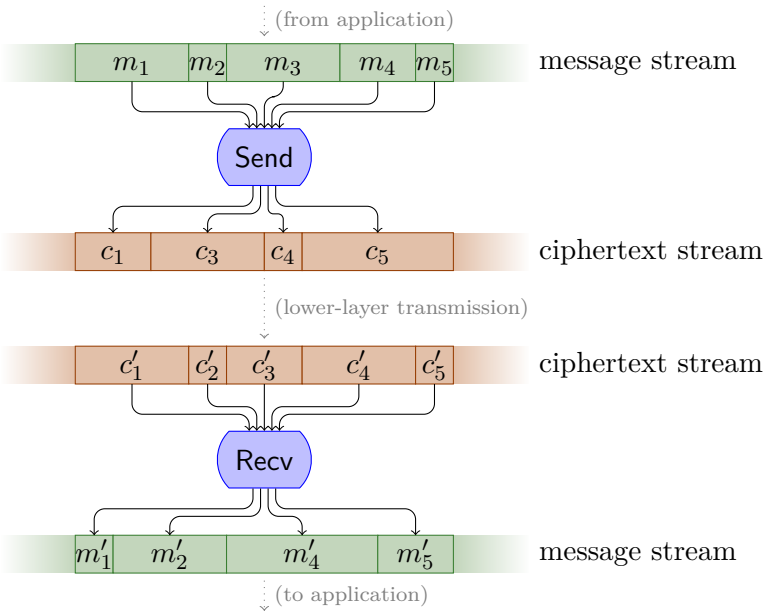


Figure 3.1: *Illustration of the behavior of the* Send *and* Recv *algorithms of a stream-based channel, indicating the message and ciphertext fragments being sent (m_i resp. c_i) and received (m'_i resp. c'_i).*

sender only processes atomic messages (i.e., each ciphertext output by Enc is the encapsulation of the input message). To enforce that particular chunks of the message stream are sent out within a specific invocation of Send we employ the established concept of 'flushing a stream' known from network socket programming, and provide the Send algorithm with an auxiliary input, namely a flush flag $f \in \{0,1\}$, for each call. If the flush flag is set to $f = 1$ then all message fragments that Send has buffered so far are processed instantaneously; otherwise, the algorithm may internally decide to accept more message fragments, to empty its buffer, or to send out only parts of it, depending on its implementation and resources. We note that our model also captures implementations that, instead of offering an explicit flushing mechanism, keep buffering their input until a specified timeout is reached. Figure 3.1 illustrates the behavior of the sending and receiving algorithms of a stream-based channel.

We proceed with defining the syntax and the correctness condition of stream-based channels.

Definition 7 (Stream-Based Channels)**.** *A* stream-based channel $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *with state space $\mathcal{S}$ and error space $\mathcal{E}$, $\mathcal{E} \cap \{0,1\}^* = \emptyset$, consists of three efficient probabilistic algorithms:*

- Init*. The initializtion algorithm takes as input a security parameter 1^λ and outputs initial states $st_S, st_R \in \mathcal{S}$ for the sender and the receiver, respectively. We write this as $(st_S, st_R) \leftarrow_\$ \mathsf{Init}(1^\lambda)$.*
- Send*. The sending algorithm takes as input a state $st_S \in \mathcal{S}$, a message fragment $m \in \{0,1\}^*$, and a flush flag $f \in \{0,1\}$, and outputs an updated state $st'_S \in \mathcal{S}$ and a ciphertext fragment $c \in \{0,1\}^*$. We write $(st'_S, c) \leftarrow_\$ \mathsf{Send}(st_S, m, f)$.*
- Recv*. The receiving algorithm takes as input a state $st_R \in \mathcal{S}$ and a ciphertext fragment $c \in \{0,1\}^*$, and outputs an updated state $st'_R \in \mathcal{S}$ and a fragment $m \in (\{0,1\} \cup \mathcal{E})^*$. We write $(st'_R, m) \leftarrow_\$ \mathsf{Recv}(st_R, c)$.*

Our syntax allows Send to return an empty output. This is to model the situation that no flush is requested and the input fragment is too short to be processed, thus, it is buffered

for later sending. In our definition below, for any message fragment m processed by Send we denote by c the (potentially empty) resulting ciphertext output by the algorithm. We stress that c should not be interpreted as an encapsulation of m (i.e., we do not require that c decrypts to m, as we explain later), but as a label for the output of Send on input m (and the current sending state). Similar considerations hold for Recv, which may as well buffer its input and return an empty output for the moment. Observe that our syntax lets Recv output a 'string' of message bits, error symbols from $\mathcal{E}$, or a combination of these; this is to capture the case that a portion of an input fragment c causes the receiving algorithm to return some message bits, while the rest of it is declared as invalid, leading to a string of errors. Note that in contrast to the other channel notions considered in this thesis, stream-based channels do not support associated data.[1]

Before formalizing the functionality of stream-based channels we set some notation used throughout this chapter. For a fixed state pair $(st_{S,0}, st_{R,0})$, an integer $\ell \geq 0$, and tuples of message fragments $\mathbf{m} = (m_1, \ldots, m_\ell) \in \{0,1\}^{**}$ and of flush flags $\mathbf{f} = (f_1, \ldots, f_\ell) \in \{0,1\}^*$, let $(st_S, \mathbf{c}) \leftarrow_\$ \mathsf{Send}(st_{S,0}, \mathbf{m}, \mathbf{f})$ be shorthand for the sequential execution $(st_{S,1}, c_1) \leftarrow_\$ \mathsf{Send}(st_{S,0}, m_1, f_1), \ldots, (st_{S,\ell}, c_\ell) \leftarrow_\$ \mathsf{Send}(st_{S,\ell-1}, m_\ell, f_\ell)$ with $\mathbf{c} = (c_1, \ldots, c_\ell)$ and $st_S = st_{S,\ell}$. For $\ell = 0$ we define $\mathbf{c}$ to be the empty vector and $st_{S,\ell} = st_{S,0}$ to be the initial state. We use an analogous notation for the receiver's algorithm.

Correctness requires that if Send is fed with a message stream, and later (a prefix of) the resulting ciphertext stream is processed by Recv, then no matter how the ciphertexts are fragmented at the sender's side and re-fragmented at the receiver's side (provided that the order of the bits is preserved), the returned message stream must be a prefix of the initial message stream. Moreover, when Recv consumes a full ciphertext fragment generated by a call to Send with the flush flag set to 1, the stream output by Recv should contain all the message fragments input to Send up to that call. We formalize this intuition in the definition below.

Definition 8 (Correctness of stream-based channels)**.** *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a stream-based channel. We say that* Ch *provides* correctness *if for all state pairs* $(st_{S,0}, st_{R,0}) \leftarrow_\$ \mathsf{Init}(1^\lambda)$*, all* $\ell, \ell' \in \mathbb{N}$*, all choices of the randomness for algorithms* Init*,* Send *and* Recv*, all message-fragment vectors* $\mathbf{m} \in (\{0,1\}^*)^\ell$*, all flush-flag vectors* $\mathbf{f} \in \{0,1\}^\ell$*, all output sequences* $(st_{S,\ell}, \mathbf{c}) \leftarrow_\$ \mathsf{Send}(st_{S,0}, \mathbf{m}, \mathbf{f})$*, all ciphertext-fragment vectors* $\mathbf{c}' \in (\{0,1\}^*)^{\ell'}$*, and all output sequences* $(st_{R,\ell'}, \mathbf{m}') \leftarrow_\$ \mathsf{Recv}(st_{R,0}, \mathbf{c}')$*, we have for every* $1 \leq i \leq \ell : f_i = 1$ *that*

$$\|\mathbf{c}[1, \ldots, i] \preccurlyeq \|\mathbf{c}' \preccurlyeq \|\mathbf{c} \implies \|\mathbf{m}[1, \ldots, i] \preccurlyeq \|\mathbf{m}' \preccurlyeq \|\mathbf{m} \ .$$

Our interpretation of correctness for stream-based channels is as follows. If the receiver obtains (in an arbitrarily fragmented way) a prefix $\|\mathbf{c}'$ of the string $\|\mathbf{c}$ of ciphertexts created by the sender for an input-message vector $\mathbf{m} = (m_1, \ldots, m_\ell)$ and flush-flag vector $\mathbf{f} = (f_1, \ldots, f_\ell)$, and if the string $\|\mathbf{c}'$ contains the concatenation $c_1 \| \ldots \| c_i$ of the first i ciphertexts of $\mathbf{c}$, then the message string $\|\mathbf{m}'$ returned on the receiver's side contains as a prefix the concatenation $m_1 \| \ldots \| m_i$. In particular, if the last sending invocation flushed its input then the receiver outputs the full genuine message stream.

Remark 1. Let us compare our correctness condition for stream-based channel with that defined by Boldyreva *et al.* [BDPS12] for symmetric encryption supporting ciphertext fragmentation (abbreviated as FrSE throughout the thesis). There, correctness requires that if a sequence $\mathbf{m}$ of discrete messages is encrypted, and the resulting ciphertext stream $\|\mathbf{c}$ is then decrypted (possibly in a fragmented manner), then the obtained message sequence (when message separators ¶ are removed) is identical to the original sequence $\mathbf{m}$. In the special case of a single message, this

[1] Indeed, the concept of associated data seems to be meaningful only in the atomic setting.

implies that encryption 'always flushes' in the setting of [BDPS12], and is in turn the reason why encryption is necessarily an atomic operation. By contrast, in our setting the Send algorithm is equipped with a flush flag and, when the latter is set to zero, potentially the entire message fragment is buffered for delayed sending. This is, then, an essential difference between the setting of Boldyreva *et al.* [BDPS12] and ours. An additional difference is that the correctness condition in [BDPS12] is technically stronger than ours as it incorporates a certain amount of 'robustness'. More specifically, the sequence of ciphertext fragments $\mathbf{c}'$ submitted for decryption in the correctness definition of [BDPS12] may extend the sequence produced by encryption, i.e., $\|\mathbf{c}$ is only required to be a prefix of $\|\mathbf{c}'$ for decryption to still work correctly up to $\|\mathbf{c}$. We find such a correctness requirement artificially strong as it explicitly demands correct decryption also in case of an active attack (which appends non-genuine fragments to a genuine stream).

3.3 Defining Security for Stream-Based Channels

We proceed with formalizing the security properties to be expected from a stream-based channel. For this we closely follow the modeling strategy that Bellare, Kohno, and Namprempre [BKN02] (BKN) introduced for analyzing the security of SSH. Since BKN see the SSH channel protocol as a stateful authenticated encryption scheme, their model is applicable to channels that offer an atomic interface. When reworking the confidentiality and integrity notions from [BKN02] to the streaming setting we are instead faced with the fact that stream-based channels support processing of arbitrary message and ciphertext fragments rather than *atomic* messages and ciphertexts. Prior to this work, ciphertext fragmentation has been considered by Boldyreva *et al.* [BDPS12] (BDPS). However, we stress that BDPS insist in letting the encryption algorithm only process atomic messages, hence ignoring the fragmentation at the sender.

3.3.1 Confidentiality

Following the general approach of BKN, we model confidentiality of stream-based channels via an indistinguishability game where an adversary $\mathcal{A}$ interacts with the sending and the receiving procedures of a channel instance through oracles.

Confidentiality against passive adversaries. To define security against passive attacks (a.k.a. *chosen plaintext-fragment attacks*, IND-CPFA) we follow the common left-or-right approach. More specifically, we let $\mathcal{A}$ query a left-or-right sending oracle $\mathcal{O}_{\mathsf{LoR}}$ on any pair (m_0, m_1) of message fragments with $|m_0| = |m_1|$, together with a flush flag f; the latter allows the adversary to control the fragmentation at the sender to the same extent a honest user does. The oracle replies each query with the ciphertext c output by Send on input the so-called challenge-message fragment m_b, the flush flag f, and the current sending state. As usual, the goal of the adversary is to find out the secret bit b. We note that the presence of a flush flag is a direct consequence of our syntax and, in fact, represents the distinguishing (but non crucial) element between atomic security (see Section 2.3 on page 10) and ours.

Defining confidentiality against active adversaries. Defining indistinguishability against active attacks (a.k.a. *chosen ciphertext-fragment attacks*, IND-CCFA) turns out to be more challenging. In addition to a left-or-right oracle, here we grant decryption capabilities to the adversary through a receiving oracle $\mathcal{O}^*_{\mathsf{Recv}}$ which $\mathcal{A}$ can query on any ciphertext fragment of her choice. The working principle of $\mathcal{O}^*_{\mathsf{Recv}}$ is based on the synchronization strategy introduced by BKN and later refined by BDPS—which essentially suppresses the decryptions of in-sync queries as they would coincide with challenge messages by correctness.

Recall that in the confidentiality experiment for stateful symmetric encryption (from Section 2.3) decryption queries are considered in-sync as long as the sequence of received ciphertexts matches the sequence of sent ciphertexts, while they are out-of-sync starting with the first decryption query that deviate from the genuine sequence. In case of symmetric encryption supporting fragmentation (FrSE) [BDPS12] instead the first deviating ciphertext fragment may be *not entirely* out-of-sync, as it may contain (or complete) genuine ciphertexts that would be decrypted correctly.[2] For this reason, the corresponding confidentiality experiment—'indistinguishability under a chosen-fragment attack', IND-CFA—identifies the point where synchronization is lost *within* the deviating fragment. Precisely, this point concides with the boundary of the last genuine ciphertext that is received entirely, since correctness is guaranteed up to this boundary (see [BDPS12]). We stress that for FrSE messages and ciphertexts correspond one-to-one at the sender, thus, by comparing the sequence of received ciphertext fragments with the sequence of sent ciphertexts, it is immediate to identify in-sync fragments and suppress the corresponding (atomic) messages. In stream-based channels, however, this one-to-one relation between sent messages and sent ciphertexts is lost.

In the IND-CCFA experiment for stream-based channels we adopt a suppression strategy similar to that of BKN and BDPS. We let $\mathcal{O}^*_{\mathsf{Recv}}$ answer each receiving query by either returning the entire output of Recv, by returning only a portion of it, or by suppressing all of it, depending on whether the received ciphertext stream is in-sync with the genuine stream or not. More formally, the experiment maintains strings C_S and C_R to store (the concatenation of) all sent and received ciphertext fragments, respectively, and a flag active that is set to 1 if an active measure of the adversary takes place. In line with BKN and BDPS, our experiment declares the attacker to become active in the moment it causes the received stream C_R to deviate from the sent stream C_S (in symbols: $C_R \not\preccurlyeq C_S$). In this case we also say that synchronization has been lost, or that C_R is out-of-sync. Correspondingly, we say that a query c to $\mathcal{O}^*_{\mathsf{Recv}}$ is out-of-sync if $C_R \,\|\, c \not\preccurlyeq C_S$, otherwise we say that it is in-sync. Clearly, if c is in-sync we are under the correctness regime and suppress the entire output of Recv that results from c. If c is out-of-sync and synchronization has been lost already, then we give the full output of Recv to the adversary. The interesting case appears when C_R *goes out-of-sync* with c, i.e., $C_R \preccurlyeq C_S$ and $C_R \,\|\, c \not\preccurlyeq C_S$. We are now faced with the question: at which point exactly shall we declare synchronization to be lost? Observe that in the streaming setting we cannot go for the choice of BDPS and lose synchronization at the ciphertext boundaries: as the output of Send is not atomic the concept of 'ciphertext boundary' does not make sense, as we clarify in the following remark.

Remark 2 (BDPS security is too strong for TLS). As stated in the TLS specification [DR08], the encryption routine that the TLS Record Protocol uses internally processes at most 2^{14} bytes per invocation. Consider an implementation where the sending procedure always flushes its input (this is indeed the case for the OpenSSL implementation). If we would instruct $\mathcal{O}^*_{\mathsf{Recv}}$ to suppress the output of Recv up to the last genuine 'ciphertext boundary' (as for BDPS), the channel would be vulnerable to the following chosen ciphertext-fragment attack, no matter how secure the underlying encryption primitive is. The adversary first asks a left-or-right query on a pair of messages (m_0, m_1), both of length at least $2^{14} + 1$ bytes and such that m_b has as first bit b, and gets back a ciphertext c; then $\mathcal{A}$ flips the last bit of c, submits the resulting ciphertext c' for decryption, and obtains a message fragment m'. If the first bit of m' is 0 then the adversary outputs 0, otherwise it outputs 1. Given that the payload of a single TLS record may not exceed 2^{14} bytes, Send is forced to output a ciphertext fragment c encapsulating

[2] Note that for FrSE supporting fragmentation we refer to the output of the encryption algorithm as a sequence of 'ciphertexts' and we name the input to the decryption algorithm a sequence of 'ciphertext fragments'; this is to highlight that the encryption algorithm outputs *atomic* entities ciphertexts while the decryption algorithm takes as input *fragments*.

(at least) *two* TLS records c_1 and c_2. Nevertheless, c is the output of a single invocation of Send and hence is deemed an 'atomic ciphertext' according to BDPS. As a consequence, c' is declared fully out of sync and hence the entire underlying message m' is given to the adversary. Now, since c' spans over both the first, unmodified TLS record c_1 and a second, non-genuine record $\overline{c_2}$, by correctness m' contains at least the full message-fragment underlying c_1, which reveals a non-empty prefix of m_b.

In the rest of this section we first discuss two candidate suppression strategies and then present our model. The natural choice in the streaming setting seems to declare synchronization to be lost with the first bit of C_R that deviates from C_S. Let C_S and C_R denote the streams of ciphertexts sent and received after $\mathcal{O}^*_{\mathsf{Recv}}$ processed in-sync queries and let c be the first out-of-sync ciphertext fragment. Let $\widetilde{c}$ be the longest prefix of c such that $C_R \,\|\, \widetilde{c} \preccurlyeq C_S$ and let c' be such that $c = \widetilde{c} \,\|\, c'$. Then we consider synchronization to be lost starting with c'. We are left with the question: which portion of the output of Recv on input c shall be suppressed by the oracle? One option would be to split the queried fragment c into its in-sync part $\widetilde{c}$ and its out-of-sync part c', process sequentially $\widetilde{c}$ and c' through Recv, and only return the output of the second invocation to the adversary. This approach, however, is sound only if the output of Recv is independent on the fragmentation of the input, which is true only for a very restricted class of channel protocols.

For a second option, we consider a suppression strategy based on the following observation: if the channel is confidential against chosen ciphertext-fragment attacks then any prefix of m that matches the corresponding challenge message-fragment m_b likely originates from the genuine prefix $\widetilde{c}$ of c. Thus, to rule out trivial wins it suffices to polish m from any bit of the challenge-message fragment before giving it to the adversary. Concretely, the idea would be that $\mathcal{O}^*_{\mathsf{Recv}}$ removes from m any common prefix with the message fragment $\widetilde{m}$ that Recv would output on $\widetilde{c}$, and returns only the message fragment $m' = m \,\%\, [m, \widetilde{m}]$ (containing the suffix of m that does not match $\widetilde{m}$) to the adversary. This should ensure that any potential challenge bit of m that comes from the genuine part of c is suppressed.

The suppression strategy just described is the one proposed in [FGMP15]. Unfortunately, without further modification, the resulting confidentiality notion fails to exclude a class of trivial attacks,[3] as we explain next.

Remark 3 (The IND-CCFA notion from [FGMP15] is too strong). Consider a chosen ciphertext-fragment attack that proceeds as follows. The adversary starts with asking a decryption query on a bit d chosen uniformly at random (that, for any reasonable scheme, results in an empty plaintext output by Recv). Note that, as no sending query has been made, this first decryption query is out-of-sync. The adversary then asks a left-or-right query $(m_0, m_1, 1)$ with $m_0 \neq m_1$ and, if the first bit of the resulting ciphertext c coincides with d, i.e., $d \preccurlyeq c$, it asks for decryption of the fragment $c \,\%\, d$, gets back a message m, and finally returns 0 if $m = m_0$ and 1 if $m = m_1$. Otherwise, if its guess was wrong and $d \not\preccurlyeq c$, it halts with output d. Note that if $d \preccurlyeq c$ we have $m = m_b$ by correctness, so $\mathcal{A}$ wins with non-negligible probability.

We stress that the attack above renders all schemes insecure according to the IND-CCFA notion from [FGMP15], and effectively applies also to the IND-CFA notion of [BDPS12], meaning that these security properties are not achievable. The problem originates from stopping the suppression process when the stream of received ciphertext only *extends* the stream of sent ciphertexts, even if no actual deviation from the sent ciphertext stream occurs. As the adversary can guess a short prefix of the next challenge ciphertext with constant probability, the game

[3]The issue was discovered by Jean Paul Degabriele, one of the authors of [BDPS12], who found the flaw applicable to the BDPS's syncing strategy, leading to a confidentiality notion (IND-CFA) that is not achievable. The confidentiality notion from [FGMP15] inherits the same flaw.

declares synchronization to be lost 'too early'. The third option is based on the above, but excludes the attack illustrated in Remark 3.

Before giving the details of the third option, the one we actually adopt, let us pause on why a similar problem does not arise in the context of stateful encryption. Note that the standard (i.e., atomic) IND-CCA notion also declares synchronization to be lost in case the adversary causes the sequence of received ciphertexts to extend the sequence of sent ciphertexts; this happens if $\mathcal{A}$ submits to $\mathcal{O}_{\mathsf{Dec}}$ all the ciphertexts output by the left-or-right oracle, in correct sequential order, and then asks for more decryptions (see line 12 of Figure 2.2, on page 14). However, guessing an entire valid ciphertext is much harder than guessing just a few bits of it. Clearly, if an adversary can a priori produce the challenge ciphertext c that the left-or-right oracle will output before the corresponding left-or-right query (m_0, m_1) is made, it should get credit for this and obtain the decryption of that ciphertext. This observation is the idea behind how we salvage the experiments of [FGMP15] and obtain a sound notion.

The IND-CCFA experiment proposed in this thesis is similar to the one from [FGMP15] except for a small but crucial tweak of the $\mathcal{O}^*_{\mathsf{Recv}}$ oracle that modifies the suppression mechanism. Namely, if $\mathcal{A}$ submits for decryption more ciphertext bits than those output by $\mathcal{O}_{\mathsf{LoR}}$, causing the stream of received ciphertext fragments to extend the stream of sent ciphertexts, i.e., $C_S \prec C_R$, then $\mathcal{O}^*_{\mathsf{Recv}}$ returns the (potentially 'polished') output of Recv but does not necessarily declare synchronization to be lost. It does if the part of C_R exceeding C_S results in a non-empty output of Recv. This allows $\mathcal{O}^*_{\mathsf{Recv}}$ to later suppress further message bits in case, as a consequence of the adversary posing more left-or-right queries, C_S and C_R end up being matching again (in the sense that $C_R \preccurlyeq C_S$). Intuitively, the idea behind our choice is to give credit to the adversary, and hence declaring it active ($\mathsf{active} \leftarrow 1$) and switching off the suppression mechanism forever, only if it produces a ciphertext fragment c that either deviates from, or exceeds, the stream of sent ciphertexts in such a way that its deviating or exceeding part $c' = c \,\%\, \widetilde{c}$ leads Recv to return a non-empty output.

In Figure 3.2 we specify the experiments of *indistinguishability under chosen plaintext-fragment attacks* (IND-CPFA) and under *chosen ciphertext-fragment attacks* (IND-CCFA) for stream-based channels. Based on the experiments, we define security as follows.

Definition 9 (Indistinguishability for stream-based channels)**.** *For* $\text{ATK} \in \{\text{CPFA}, \text{CCFA}\}$ *we say that a streaming channel* Ch *offers* ATK-indistinguishability *if for all efficient adversaries* $\mathcal{A}$ *the following advantage function is negligible,*

$$\mathbf{Adv}^{\text{IND-ATK}}_{\mathsf{Ch},\mathcal{A}}(\lambda) := \left|\Pr\left[\text{Expt}^{\text{IND-ATK},1}_{\mathsf{Ch},\mathcal{A}}(1^\lambda) = 1\right] - \Pr\left[\text{Expt}^{\text{IND-ATK},0}_{\mathsf{Ch},\mathcal{A}}(1^\lambda) = 1\right]\right| \ .$$

*We abbreviate the notions of indistinguishability under chosen plaintext-fragment attacks (*CPFA*-indistinguishability) and of indistinguishability under chosen ciphertext-fragment attacks (*CCFA*-indistinguishability) for streaming channels by writing IND-CPFA and IND-CCFA, respectively.*

3.3.2 Integrity

We proceed with defining integrity notions for stream-based channels. While integrity in the atomic setting (stateless and stateful authenticated encryption, e.g., [BN00, BKN02]) is well-understood, no previous work considered integrity in the presence of fragmentation. In particular, for symmetric encryption supporting ciphertext fragmentation, Boldyreva *et al.* [BDPS12] only address confidentiality. Again we formalize security via games involving an adversary $\mathcal{A}$ that interacts with sending and receiving algorithms through oracles.

Plaintext-stream integrity. Intuitively, the security property of *plaintext-stream integrity* (INT-PST) says that it is difficult to make Recv output a message stream that is not a prefix

$\mathrm{Expt}^{\text{IND-ATK},b}_{\mathsf{Ch},\mathcal{A}}(1^\lambda)$
01 $(st_S, st_R) \leftarrow_\$ \mathsf{Init}(1^\lambda)$
02 $\mathsf{active} \leftarrow 0$
03 $C_S \leftarrow \varepsilon$, $C_R \leftarrow \varepsilon$
04 $b' \leftarrow_\$ \mathcal{A}(1^\lambda)^{\mathcal{O}_{\mathsf{LoR}}, \mathcal{O}^*_{\mathsf{Recv}}}$
05 Terminate with b'

$\mathcal{O}_{\mathsf{LoR}}(m_0, m_1, f)$:
06 Require $|m_0| = |m_1|$
07 $(st_S, c) \leftarrow_\$ \mathsf{Send}(st_S, m_b, f)$
08 $C_S \leftarrow C_S \,\|\, c$
09 Return c to $\mathcal{A}$

ATK = CCFA
$\mathcal{O}^*_{\mathsf{Recv}}(c)$:
10 If $\mathsf{active} = 1$:
11 $(st_R, m) \leftarrow_\$ \mathsf{Recv}(st_R, c)$
12 Return m to $\mathcal{A}$
13 Else if $C_R \,\|\, c \preccurlyeq C_S$:
14 $(st_R, m) \leftarrow_\$ \mathsf{Recv}(st_R, c)$
15 $C_R \leftarrow C_R \,\|\, c$
16 Return ε to $\mathcal{A}$
17 Else:
18 If $C_R \prec [C_R \,\|\, c, C_S]$:
19 $\widetilde{c} \leftarrow [C_R \,\|\, c, C_S] \,\%\, C_R$
20 $\widetilde{st_R} \leftarrow st_R$
21 $(\widetilde{st_R}, \widetilde{m}) \leftarrow_\$ \mathsf{Recv}(\widetilde{st_R}, \widetilde{c})$
22 $(st_R, m) \leftarrow_\$ \mathsf{Recv}(st_R, c)$
23 $m' \leftarrow m \,\%\, [m, \widetilde{m}]$
24 Else:
25 $(st_R, m') \leftarrow_\$ \mathsf{Recv}(st_R, c)$
26 If $C_S \not\preccurlyeq C_R \,\|\, c$ or $m' \neq \varepsilon$:
27 $\mathsf{active} \leftarrow 1$
28 Else:
29 $C_R \leftarrow C_R \,\|\, c$
30 Return m' to $\mathcal{A}$

Figure 3.2: *Indistinguishability experiments for stream-based channels. We assume $m_0, m_1, c \in \{0,1\}^*$ and $f \in \{0,1\}$ for all such values provided by the adversary. In the IND-CPFA experiment the adversary has access to the oracle $\mathcal{O}_{\mathsf{LoR}}$ only.*

of the message stream originally input to Send. Our experiment provides $\mathcal{A}$ with a sending oracle $\mathcal{O}_{\mathsf{Send}}$ that, upon being queried on any message fragment m and flush flag f, returns the output of Send on these inputs, as well as a receiving oracle $\mathcal{O}_{\mathsf{Recv}}$ which returns on input any ciphertext fragment c the corresponding output of Recv. The game maintains strings M_S and M_R to store (the concatenation of) the sequence of message fragments queried to $\mathcal{O}_{\mathsf{Send}}$ and of the sequence of fragments returned by $\mathcal{O}_{\mathsf{Recv}}$, respectively. The adversary wins the game as soon as M_R deviates from M_S and the deviation contains some message bits (beyond error symbols).

Ciphertext-stream integrity. Intuitively, from *ciphertext-stream integrity* (INT-CST) we expect that not only the message stream be transmitted without modification but also that the ciphertext stream cannot be manipulated without detection. This is, clearly, a stronger requirement than plaintext-stream integrity. The setup here is as for the INT-PST experiment except for replacing the message strings M_S and M_R with the ciphertext strings C_S and C_R, for the sent and received ciphertext streams respectively. Our experiment reflects the intuition that, when processing an out-of-sync ciphertext fragment, the receiving algorithm should return an error. Formalizing this intuition requires some care. Indeed, for a streaming interface it may happen that Recv processes a ciphertext fragment which does not yet contain 'enough information' to be verified; in this case the receiving algorithm would buffer that fragment and

$\mathrm{Expt}^{\text{INT-ATK}}_{\mathsf{Ch},\mathcal{A}}(1^\lambda)$
01 $(st_S, st_R) \leftarrow_\$ \mathsf{Init}(1^\lambda)$
02 $\mathsf{active} \leftarrow 0$
03 $M_S \leftarrow \varepsilon$, $M_R \leftarrow \varepsilon$
04 $C_S \leftarrow \varepsilon$, $C_R \leftarrow \varepsilon$
05 $\mathcal{A}(1^\lambda)^{\mathcal{O}_{\mathsf{Send}},\mathcal{O}_{\mathsf{Recv}}}$
06 Terminate with 0

$\mathcal{O}_{\mathsf{Send}}(m, f)$:
07 $(st_S, c) \leftarrow_\$ \mathsf{Send}(st_S, m, f)$
08 $M_S \leftarrow M_S \,\|\, m$
09 $C_S \leftarrow C_S \,\|\, c$
10 Return c to $\mathcal{A}$

ATK = PST
$\mathcal{O}_{\mathsf{Recv}}(c)$:
11 $(st_R, m) \leftarrow_\$ \mathsf{Recv}(st_R, c)$
12 $M_R \leftarrow M_R \,\|\, m$
13 If $M_R \,\%\, [M_R, M_S] \notin \mathcal{E}^*$:
14 Terminate with 1
15 Return m to $\mathcal{A}$

ATK = CST
$\mathcal{O}_{\mathsf{Recv}}(c)$:
16 If $\mathsf{active} = 1$:
17 $(st_R, m) \leftarrow_\$ \mathsf{Recv}(st_R, c)$
18 If $m \notin \mathcal{E}^*$: Terminate with 1
19 Else if $C_R \,\|\, c \preccurlyeq C_S$:
20 $(st_R, m) \leftarrow_\$ \mathsf{Recv}(st_R, c)$
21 $C_R \leftarrow C_R \,\|\, c$
22 Return m to $\mathcal{A}$
23 Else:
24 If $C_R \prec [C_R \,\|\, c, C_S]$:
25 $\widetilde{c} \leftarrow C_S \,\%\, C_R$
26 $\widetilde{st_R} \leftarrow st_R$
27 $(\widetilde{st_R}, \widetilde{m}) \leftarrow_\$ \mathsf{Recv}(\widetilde{st_R}, \widetilde{c})$
28 $(st_R, m) \leftarrow_\$ \mathsf{Recv}(st_R, c)$
29 $m' \leftarrow m \,\%\, [m, \widetilde{m}]$
30 Else:
31 $(st_R, m') \leftarrow_\$ \mathsf{Recv}(st_R, c)$
32 $m \leftarrow m'$
33 If $C_S \not\preccurlyeq C_R \,\|\, c$ or $m' \neq \varepsilon$:
34 $\mathsf{active} \leftarrow 1$
35 Else:
36 $C_R \leftarrow C_R \,\|\, c$
37 If $m' \notin \mathcal{E}^*$: Terminate with 1
38 Return m to $\mathcal{A}$

Figure 3.3: *Integrity experiments for stream-based channels. We assume $m, c \in \{0,1\}^*$ and $f \in \{0,1\}$ for all such values provided by the adversary.*

wait for further input until a sufficiently long string is available. In such a scenario, a naive adaptation of the ciphertext integrity definition of [BKN02] (INT-CTXT, see Section 2.3 on page 10) would allow trivial attacks by declaring any adversary successful that makes Recv buffer (part of) an out-of-sync ciphertext. Our experiment identifies the case just described and consistently lets $\mathcal{O}_{\mathsf{Recv}}$ wait for further ciphertext fragments in case Recv produces no output when processing an out-of-sync fragment; it only declares the adversary successful if Recv outputs a *non-empty* message fragment resulting from a deviating or exceeding portion of the ciphertext stream. This is in line with our IND-CCFA confidentiality notion, declaring the adversary active if it produces a deviation or an extension of the stream C_S that yields a non-empty message string.

In Figure 3.3 we specify the experiments of *integrity of plaintext streams* (INT-PST) and of *ciphertext streams* (INT-CST) for stream-based channels. Security is defined below.

Definition 10 (Integrity for stream-based channels)**.** *For* ATK $\in \{\text{PST}, \text{CST}\}$ *we say that a stream-based channel* Ch *offers* ATK-integrity *if for all efficient adversaries $\mathcal{A}$ the following advantage function is negligible,*

$$\mathbf{Adv}^{\text{INT-ATK}}_{\mathsf{Ch},\mathcal{A}}(\lambda) := \left|\Pr\left[\mathrm{Expt}^{\text{INT-ATK}}_{\mathsf{Ch},\mathcal{A}}(1^\lambda) = 1\right]\right| \quad .$$

*We abbreviate the notions of integrity of plaintext streams (*PST*-integrity) and of integrity of ciphertext streams (*CST*-integrity) by writing INT-PST and INT-CST, respectively.*

3.3.3 Evaluation of Our Security Notions

When formalizing a security goal, a common challenge in cryptography is to develop a notion that is simultaneously as strong as possible and achievable. On the one hand, we would like a security model to account for the largest class of potential adversaries. On the other hand, we also would like that reasonable schemes exist which are secure in this model. What 'potential adversary' and 'reasonable scheme' mean depends on the application that is to be cryptographically protected. The stronger the model, the fewer the schemes which are secure. As shown by the number of schemes that were proven secure on paper but turned out to be vulnerable in practice, designing models that take into account a large class of attacks is highly critical. The inability of foreseeing new attacks is the main reason for aiming at the strongest (achievable) notion of security.

Ideally we would also like to prevent that our model errs in the opposite direction by declaring insecure schemes that are, in fact, intuitively secure.[4] If this happens, we say that the security notion is too strong. All in all, a cryptographer is left with the difficult task of finding a trade-off between making the model sufficiently strong to capture realistic attacks while keeping it weak enough in order not to exclude good schemes.

As an example of a security model that is (arguably) too strong consider the IND-CCA notion, understood by most researchers as 'the right' confidentiality goal for symmetric encryption. This notion allows the adversary to see the decryption of any chosen ciphertext except for challenge ciphertexts (or in-sync ciphertexts in the stateful setting), i.e., ciphertexts that decrypt to challenge messages by correctness. Indeed, obtaining the decryption of challenge ciphertexts would let the adversary win the distinguishing game in a trivial way and, thus, render the corresponding notion unreasonably strong. The IND-CCA model is, to some extent, too strong for some applications, in the sense that we explain below. There exist encryption schemes which are intuitively secure but provably IND-CCA-insecure. For instance, if one starts with an IND-CCA-secure encryption scheme and modifies it by letting the encryption routine append a redundant bit to each ciphertext and letting the decryption routine ignore the last bit of each ciphertext, the resulting scheme is no longer IND-CCA-secure. However, adding a redundant bit that is then ignored for decryption should not cause any harm.[5] Put differently, the IND-CCA notion allows for 'trivial attacks', namely attacks that are possible within the model because of some artificial capabilities the adversary is given in the security experiment (here, flipping a bit and getting the decryption of a ciphertext that by design decrypts to the challenge message). Our IND-CCFA notion for stream-based channel is likewise too strong for some applications, as we show next.

An intuitively confidential stream-based channel. In July 2016 the authors of [FGMP15] were informed about the existence of a class of stream-based channels that are intuitively confidential but deemed insecure according to the IND-CCFA model.[6] The channel uses an AEAD scheme as a building block and works as follows. Send chops the input message fragment m

[4] Of course, claims about the 'intuitive security' of a scheme have no formal ground and cryptographers should be careful not to blindly follow one's intuition, which may be misleading. However, behind a security notion there is always one's intuition regarding the properties that a scheme should (or should not) offer.

[5] An alternative notion of confidentiality that precisely aims at resolving this issue, called RCCA security (for 'replayable' CCA), was proposed by Canetti *et al.* in [CKN03].

[6] The scheme depicted in Figure 3.2 was proposed by Bertram Poettering during the second workshop on 'Secure Key Exchange and Channel Protocols' (SKECH2), Bertinoro, Italy.

$\mathsf{Init}(1^\lambda)$
01 $K \leftarrow_\$ \mathsf{KeyGen}(1^\lambda)$
02 $st_{S,0} \leftarrow (K, 0)$
03 $st_{R,0} \leftarrow (K, 0, \varepsilon, 0)$
04 Return $(st_{S,0}, st_{R,0})$

$\mathsf{Send}(st_S, m, f)$
05 Parse st_S as (K, ctr)
06 $l \leftarrow |m|$
07 $m_1 \dots m_l \leftarrow m$
08 $c \leftarrow \varepsilon$
09 For $j \leftarrow 1$ to l:
10 $\quad c_j \leftarrow \mathsf{Enc}_K(\mathrm{ctr}, m_j)$
11 $\quad c \leftarrow c \| c_j$
12 $\mathrm{ctr} \leftarrow \mathrm{ctr} + 1$
13 $st_S \leftarrow (K, \mathrm{ctr})$
14 Return (st_S, c)

$\mathsf{Recv}(st_R, c)$
15 Parse st_R as $(K, \mathrm{ctr}, buf, \mathsf{fail})$
16 $buf \leftarrow buf \| c$
17 $l \leftarrow \lfloor |buf|/clen \rfloor$
18 $c_1 \dots c_l \| buf \leftarrow buf$
19 $m \leftarrow \varepsilon$
20 For $j \leftarrow 1$ to l:
21 $\quad m_j \leftarrow \mathsf{Dec}_K(\mathrm{ctr}, c_j)$
22 $\quad$ If $m_j = \perp$: $\mathsf{fail} \leftarrow 1$
23 $\quad m \leftarrow m \| m_j$
24 $\quad \mathrm{ctr} \leftarrow \mathrm{ctr} + 1$
25 If $\mathsf{fail} = 1$: $m \leftarrow 0^l$
26 $st_R \leftarrow (K, \mathrm{ctr}, buf, \mathsf{fail})$
27 Return (st_R, m)

Figure 3.4: *A stream-based channel* $\mathsf{Ch}' = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *that uses an AEAD scheme* $AEAD = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *as a building block. In line 07 we assume* $|m_j| = 1$ *and in lines 10 and 18 we assume* $|c_j| = clen$*, for all* $1 \le j \le l$ *and some constant* $clen$*. As we explain in this section,* Ch' *is intuitively a confidential channel but is declared insecure according to our IND-CCFA notion.*

into fixed-length blocks $m_1 \dots m_l$ (with $|m_i| = 1$ for simplicity), AEAD encrypts these blocks sequentially using a counter as associated data, and outputs the concatenation $c = c_1 \| \cdots \| c_l$ of the corresponding ciphertexts (which also have fixed length, $|c_i| = clen$); Recv appends the input ciphertext fragment c' to its buffer buf, extracts from the updated buffer the longest sequence of ciphertext blocks $c_1 \dots c_l$ and keeps the remaining fragment in the buffer, AEAD decrypts each block c_i and outputs the concatenation of the resulting message blocks $m_1 \| \cdots \| m_l$ as long as no AEAD decryption fails, otherwise it returns an l-bit string of 0s. We name this scheme Ch' and give full details in Figure 3.4.

By construction, Recv never outputs an error and hence Ch' provides no integrity protection. This, however, should not harm confidentiality. Indeed, by confidentiality of the AEAD scheme, the left-or-right sending oracle should reveal no information about the challenge messages, while AEAD integrity should guarantee that the receiving oracle only returns a string of 0s when queried on out-of-sync ciphertext fragments. Thus, we expect Ch' to provide confidentiality against chosen ciphertext-fragment attacks (IND-CCFA).

Confidentiality attack against Ch'. The stream-based channel Ch' is not confidential in the IND-CCFA sense. Consider an adversary $\mathcal{A}$ that proceeds as follows: it chooses $m_0 = 00, m_1 = 10$ and queries $c \leftarrow_\$ \mathcal{O}_{\mathsf{LoR}}(m_0, m_1, 0)$. Let $c = c_1 c_2$ with $|c_1| = |c_2| = clen$. The adversary then derives $c' = c_1 \overline{c_2}$ from c by inverting the bits of c_2 but leaving c_1 unmodified, and requests the decryption $m' \leftarrow_\$ \mathcal{O}^*_{\mathsf{Recv}}(c')$. Within the $\mathcal{O}^*_{\mathsf{Recv}}$ oracle we obtain $\mathsf{sync} = 0$, $\widetilde{c} = c_1$, $\widetilde{m} = b$, and $m = 00$. In particular, if $b = 0$ we have $[m, \widetilde{m}] = 0$ and thus $m' = 0$, and if $b = 1$ we have $[m, \widetilde{m}] = \varepsilon$ and thus $m' = 00$. The adversary outputs $b' = |m'| - 1$ and achieves a distinguishing advantage of 1.

Discussion. The receiving algorithm of channel Ch' returns a valid message fragment even if the AEAD decryption internally rejects, in which case it outputs a string of 0s. On the one hand, this channel construction is artificial as it explicitly hides decryption errors, neglecting

integrity. On the other hand, it does constitute a reasonable scheme in a setting where only confidentiality does matter. Thus, while Ch' lacks integrity protection and hence does not provide a 'proper' secure channel, it has no actual vulnerability in terms of confidentiality and should be treated as such. This highlights that our IND-CCFA notion excludes some schemes and might be considered too strong.

Which part of IND-CCFA is responsible for this? Why does Ch' fall prey of the above attack? As we saw, AEAD security should ensure that the decryption m of the out-of-sync ciphertext $c' = c_1\overline{c_2}$ reveals nothing about the challenge message m_b. However, within the IND-CCFA experiment, when answering query c' the receiving oracle does leak information about m_b. It says whether $m = 00$ has a non-empty common prefix with $\widetilde{m} = b$ or, equivalently, whether $b = 0$ or not. So, while m does not say anything about m_b, the oracle $\mathcal{O}^*_{\mathsf{Recv}}$ itself artificially does.

3.4 Relations Among Notions

In this section we explore relations between confidentiality and integrity—well-established for atomic messages by [BN00, BKN02] and follow-up work—and investigate whether these relations can be lifted to our streaming setting. We highlight that, since integrity for encryption schemes supporting ciphertext fragmentation was not addressed in [BDPS12], we are the first to consider such relations in the presence of fragmentation.

Ideally, we would like to translate the classic implications IND-CCA $\implies$ IND-CPA for confidentiality, INT-CTXT $\implies$ INT-PTXT for integrity, and the powerful composition result IND-CPA $\wedge$ INT-CTXT $\implies$ IND-CCA, all from [BN00], to the realm of stream-based channels. It is immediate to see that, as in the setting where messages are atomic, the stronger notions implies the weaker ones for both confidentiality and integrity individually. Unfortunately, when integrity and confidentiality are targeted simultaneously, the situation for streams is fundamentally more challenging.

Recall that, in the atomic-message setting, the proof of the composition theorem proceeds in two steps: starting from the IND-CCA game one first bounds the probability that the adversary submits for decryption an out-of-sync ciphertext that turns out to be valid by using the INT-CTXT advantage. This then allows a reduction to the IND-CPA experiment (now assuming integrity of ciphertexts), simply by answering all decryption queries with the distinguished error symbol $\perp$.[7] As already noted by Boldyreva *et al.* [BDPS14], the same proof strategy does not work for schemes that have multiple decryption error symbols (which models common real-world behavior of encryption schemes). This is because the reduction can no longer predict which one of the several possible error symbols should be output when simulating decryption.

Thus the classic result IND-CPA $\wedge$ INT-CTXT $\implies$ IND-CCA does not follow in this situation. Worse, [BDPS14] shows that, in the multiple decryption error setting, there exist schemes that are secure in both IND-CPA and INT-CTXT senses, yet are not IND-CCA secure. We show later in this section that similar issues arise for stream-based channels, even when restricting to the case of single error messages. Specifically, fragmentation at the receiver's side makes it harder to emulate a receiving oracle for the IND-CCFA experiment given a receiving oracle for the INT-CST game.

As a remedy we propose a stronger version of the composition theorem, resurrecting the result both in our streaming setting and in the case of multiple errors that was considered in [BDPS14]. However, this result can be proven only at the cost of introducing further assumptions on the output behavior of the receiving algorithm. The conditions for the composition

[7] We give an independent proof of this result in the atomic setting (but for broadcast channels, i.e., channels that support multiple participants) in Chapter 5 (on page 60).

theorem may initially look quite demanding but, as we confirm in Section 3.5 (on page 34), there exist natural schemes that satisfy the required conditions. Moreover, the use of the composition theorem is not the only route to achieving IND-CCFA security: for specific schemes it may be possible to prove IND-CCFA security directly.

Confidentiality. A study of the experiments in Figure 3.2 immediately shows that IND-CCFA security implies IND-CPFA security, since an attacker in the IND-CPFA game only needs to emulate the left-or-right oracle to provide a faithful simulation of the IND-CCFA game, and can trivially do so by relaying all encryption queries to its own left-or-right oracle.

Integrity. Assume that ciphertext-stream integrity (INT-CST) from Definition 10 holds for a stream-based channel. Then the channel also provides integrity of plaintext streams (INT-PST) and the security reduction is tight. To see why consider the integrity experiment depicted in Figure 3.3: given the INT-CST property, every efficient adversary either never produces a ciphertext stream C_R that deviates from the ciphertext stream C_S (hence, by correctness, no deviation will occur in the underlying message streams) or, if it generates a stream C_R that does deviate from C_S, by INT-CST the underlying message streams will only differ by an error string. We formalize this intuition in the proof of the following proposition.

Proposition 1. *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a correct stream-based channel which is INT-CST-secure. Then the channel is also INT-PST secure. Furthermore, for every algorithm* $\mathcal{A}$ *we have* $\mathbf{Adv}^{\text{INT-PST}}_{\mathsf{Ch},\mathcal{A}}(\lambda) \leq \mathbf{Adv}^{\text{INT-CST}}_{\mathsf{Ch},\mathcal{A}}(\lambda)$.

Proof. Assume that $\mathcal{A}$ attacks the INT-PST property of the channel. First note that $\mathcal{A}$ has the same interfaces as if attacking INT-CST such that we can think of running both experiments simultaneously. It then suffices to show that, if we trigger line 13 of the INT-PST experiment, then we would also trigger either line 18 or line 37 in the simultaneous execution of the INT-CST experiment (both in Figure 3.3 on page 26). Note that as long as $\mathsf{active} = 0$ and the ciphertext stream submitted to $\mathcal{O}_{\mathsf{Recv}}$ is a prefix of the one created by $\mathcal{O}_{\mathsf{Send}}$, then the receiver's oracle in experiment INT-CST would indeed return the recovered message fragment, as in the INT-PST experiment.

Suppose that $\mathcal{A}$ triggers line 13 in the INT-PST experiment. If at this point $C_R \preccurlyeq C_S$ then, because of correctness of the channel, we must also have $M_R \preccurlyeq M_S$, implying that lines 18 and 37 would not have been triggered. It follows that there must exist some fragment c such that $C_R \parallel c \not\preccurlyeq C_S$ and c is the first fragment queried to $\mathcal{O}_{\mathsf{Recv}}$ that causes C_R to deviate from C_S. At this point we must also have $\mathsf{active} = 1$ in the INT-CST experiment and we enter the 'else' case in line 23. Now if c contains some non-deviating prefix $\widetilde{c} \preccurlyeq c$, we compute $\widetilde{c}$ such that $C_R \parallel \widetilde{c} \preccurlyeq C_S$ is maximal. It again follows from correctness that for the processed $\widetilde{c}$ we get a message fragment $\widetilde{m}$ such that $M_R \parallel \widetilde{m} \preccurlyeq M_S$. But in order to trigger line 13 in the INT-PST experiment, we must have that the full fragment c makes Recv output a message fragment m that contains message bits beyond the common prefix with M_S. It follows that $m' \leftarrow m \,\%\, [m, \widetilde{m}]$ must contain some non-error symbols. But then we also trigger line 37 in the INT-CST experiment run.

If c contains only deviating (or exceeding) bits or if synchronization was lost before, then the full resulting message fragment (m' resp./ m) is considered for the winning condition: whenever $\mathcal{A}$ in this case wins in the INT-PST experiment, it also does in the INT-CST experiment. □

Generic composition. We already explained earlier in this section that standard arguments to prove the composition theorem do not apply in the streaming setting. The issue here is that losing the integrity game does not make the output of $\mathcal{O}^*_{\mathsf{Recv}}$ (in the confidentiality game)

$\mathrm{Expt}^{\text{ERR-PRE}}_{\mathsf{Ch},\mathcal{A}}(1^\lambda)$
01 $(st_S, st_R) \leftarrow_\$ \mathsf{Init}(1^\lambda)$
02 $C_S \leftarrow C_R \leftarrow \varepsilon$
03 $\mathcal{A}(1^\lambda)^{\mathcal{O}_{\mathsf{Send}}(\cdot,\cdot),\mathcal{O}_{\mathsf{Recv}}(\cdot)}$
04 Terminate with 0

$\mathcal{O}_{\mathsf{Send}}(m, f)$:
05 $(st_S, c) \leftarrow_\$ \mathsf{Send}(st_S, m, f)$
06 $C_S \leftarrow C_S \,\|\, c$
07 Return c to $\mathcal{A}$

$\mathcal{O}_{\mathsf{Recv}}(c)$:
08 $(st_R, m) \leftarrow_\$ \mathsf{Recv}(st_R, c)$
09 If $\mathsf{Pred}(C_S, C_R, c) \neq \langle m \rangle_{\mathcal{E}}$:
10 Terminate with 1
11 $C_R \leftarrow C_R \,\|\, c$
12 Return m to $\mathcal{A}$

Figure 3.5: *Security experiment for* error predictability *(ERR-PRE) of stream-based channels. The function* $\langle\cdot\rangle_{\mathcal{E}} \colon (\{0,1\} \cup \mathcal{E})^* \to \mathcal{E}^*$ *denotes the 'projection onto the error space' that extracts the error components from a string of symbols in* $\{0,1\}^* \cup \mathcal{E}^*$*, e.g., if* $m = 01001\bot_1\bot_2$*, then* $\langle m \rangle_{\mathcal{E}} = \bot_1\bot_2$*.*

predictable. Therefore, any strategy which allows the recovery of the composition theorem should make it possible to forecast the output behavior of the receiving algorithm when certain conditions are met. In line with this observation we introduce a new notion, so-called *error predictability*, which precisely formalizes the ability to efficiently predict (part of) the output of Recv in case an empty output or error messages are expected. Intuitively speaking, error predictability demands that error symbols (if any) returned by Recv on input any ciphertext fragment c can be efficiently predicted given the ciphertext stream C_S output by Send, the ciphertext stream C_R input to Recv, and c (i.e., given only public information).

As formalized in Definition 11 and through the security experiment of Figure 3.5, we say that a channel is *error-predictable* (ERR-PRE) if there exists an efficient algorithm Pred that, given C_R, C_S and c, outputs the above-mentioned error string, where we require the output of Pred to be accurate with high probability, for every c chosen by the adversary. Put differently, the ERR-PRE experiment declares the adversary to be successful if it ever queries a (counterfeit) ciphertext c that induces Recv to produce a (potentially empty) error string that differs from the output of the predictor.

Definition 11 (Error predictability (ERR-PRE))**.** *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a stream-based channel with error space* $\mathcal{E}$*, and let* $\mathsf{Pred} \colon \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^* \to \mathcal{E}^*$ *be an efficient deterministic algorithm. We say that* Ch *provides* error predictability with respect to the predictor Pred *if for every efficient adversary* $\mathcal{A}$ *playing the experiment defined in Figure 3.5 against channel* Ch*, the following advantage function is negligible:*

$$\mathbf{Adv}^{\text{ERR-PRE}}_{\mathsf{Ch},\mathcal{A}}(\lambda) := \Pr\left[\mathrm{Expt}^{\text{ERR-PRE}}_{\mathsf{Ch},\mathcal{A}}(1^\lambda) = 1\right] \ .$$

More generally, we say that Ch *provides error predictability (ERR-PRE) if there exists a predictor* Pred *that satisfies the above condition.*

The next theorem formalizes the idea that, for the class of error-predictable channels, the generic composition theorem holds (even for channels supporting multiple decryption errors).

Theorem 1 (IND-CPFA $\wedge$ INT-CST $\wedge$ ERR-PRE $\Longrightarrow$ IND-CCFA)**.** *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a (correct) stream-based channel with associated error space* $\mathcal{E}$*. If* Ch *provides integrity of ciphertext streams, error predictability, and indistinguishability under chosen plaintext-fragment attacks then it also provides indistinguishability under chosen ciphertext-fragment attacks. Formally, for every efficient adversary* $\mathcal{A}$ *against the IND-CCFA properties of* Ch *there exist efficient adversaries* $\mathcal{B}$*,* $\mathcal{C}$*, and* $\mathcal{D}$ *such that*

$$\mathbf{Adv}^{\text{IND-CCFA}}_{\mathsf{Ch},\mathcal{A}}(\lambda) \leq 2 \cdot \mathbf{Adv}^{\text{INT-CST}}_{\mathsf{Ch},\mathcal{B}}(\lambda) + 2 \cdot \mathbf{Adv}^{\text{ERR-PRE}}_{\mathsf{Ch},\mathcal{C}}(\lambda) + \mathbf{Adv}^{\text{IND-CPFA}}_{\mathsf{Ch},\mathcal{D}}(\lambda) \ .$$

Proof. We will consider a sequence of game transitions from the IND-CCFA experiment to the IND-CPFA experiment and bound the difference in probability between each game and its successor in the sequence with the advantage of a specific adversary. For better legibility we will denote the intermediate games by $\mathrm{E}_{\mathcal{A}}^{i,b}$ for $i \in \{0,1,2\}$ and, with a slight abuse of notation, use the shortcut $\Pr[\mathrm{E}_{\mathcal{A}}^{i,b}]$ to indicate the probability $\Pr[\mathrm{E}_{\mathsf{Ch},\mathcal{A}}^{i,b} = 1]$.

Starting from the IND-CCFA experiment of Figure 3.2 (on page 25) against $\mathcal{A}$, that we denote by $\mathrm{E}_{\mathcal{A}}^{0,b}$, we define a new experiment which penalizes the adversary in case it causes the conditions to violate ciphertext integrity of streams. More precisely, let bad_I^b be the event that the output of Recv on input a deviating or exceeding ciphertext fragment be a valid, non-empty message, i.e., $m \notin \mathcal{E}^*$ in line 11 or $m' \notin \mathcal{E}^*$ in line 26 of Figure 3.2 (on page 25). Now, let us define $\mathrm{E}_{\mathcal{A}}^{1,b}$ from $\mathrm{E}_{\mathcal{A}}^{0,b}$ by modifying the oracle $\mathcal{O}_{\mathsf{Recv}}^*$ as follows: we add extra check before lines 11 and 26 (in Figure 3.2 on page 25) and force termination of the experiment (with output value 0) in case bad_I^b occurs. Since $\mathrm{E}_{\mathcal{A}}^{1,b}$ and $\mathrm{E}_{\mathcal{A}}^{0,b}$ execute the same instructions as long as bad_I^b does not happen, we can assert that $\Pr[\mathrm{E}_{\mathcal{A}}^{0,b} \wedge \neg bad_I^b] = \Pr[\mathrm{E}_{\mathcal{A}}^{1,b} \wedge \neg bad_I^b]$, and hence obtain the bound $|\Pr[\mathrm{E}_{\mathcal{A}}^{0,b}] - \Pr[\mathrm{E}_{\mathcal{A}}^{1,b}]| \leq \Pr[bad_I^b]$.

We show next how to convert any adversary $\mathcal{A}$ that triggers either event bad_I^0 or bad_I^1 into an adversary $\mathcal{B}$ that violates the INT-CST security of Ch. Adversary $\mathcal{B}$ initially chooses a bit d uniformly at random and then runs $\mathcal{A}$, answering its queries as follows. If $\mathcal{A}$ queries (m_0, m_1, f) to $\mathcal{O}_{\mathsf{LoR}}$ then $\mathcal{B}$ queries its oracle $\mathcal{O}_{\mathsf{Send}}$ about (m_d, f), and forwards the oracle's answer to $\mathcal{A}$. Similarly $\mathcal{B}$ relays every receiving query c to its oracle $\mathcal{O}_{\mathsf{Recv}}$, obtains a response m, and returns the projection $\langle m \rangle_{\mathcal{E}}$ of m onto the error space $\mathcal{E}$ to $\mathcal{A}$. Note that if the fragment that $\mathcal{A}$ is supposed to obtain would contain any non-error symbol then it would trigger the bad event. When $\mathcal{A}$ halts, so does $\mathcal{B}$.

Observe that $\mathcal{B}$ perfectly simulates the IND-CCFA experiment as long as $\mathcal{A}$ does not trigger the event bad_I^b. Moreover, if $\mathcal{A}$ triggers bad_I^b, then $\mathcal{B}$ wins in the INT-CST experiment if it had chosen $d = b$. We can hence derive the inequality $\mathbf{Adv}_{\mathsf{Ch},\mathcal{B}}^{\text{INT-CST}}(\lambda) \geq \Pr[bad_I^0 \wedge d = 0] + \Pr[bad_I^1 \wedge d = 1] = \frac{1}{2} \cdot \Pr[bad_I^0] + \frac{1}{2} \cdot \Pr[bad_I^1]$, from which we can bound the advantage of $\mathcal{A}$ in the IND-CCFA experiment as follows:

$$\begin{aligned}
\mathbf{Adv}_{\mathsf{Ch},\mathcal{A}}^{\text{IND-CCFA}}(\lambda) &= \left|\Pr[\mathrm{E}_{\mathcal{A}}^{0,1}] - \Pr[\mathrm{E}_{\mathcal{A}}^{0,0}]\right| \\
&\leq \left|\Pr[\mathrm{E}_{\mathcal{A}}^{0,1}] - \Pr[\mathrm{E}_{\mathcal{A}}^{1,1}]\right| + \left|\Pr[\mathrm{E}_{\mathcal{A}}^{1,1}] - \Pr[\mathrm{E}_{\mathcal{A}}^{1,0}]\right| + \left|\Pr[\mathrm{E}_{\mathcal{A}}^{1,0}] - \Pr[\mathrm{E}_{\mathcal{A}}^{0,0}]\right| \\
&\leq \Pr[bad_I^1] + \left|\Pr[\mathrm{E}_{\mathcal{A}}^{1,1}] - \Pr[\mathrm{E}_{1,0}^{\mathcal{A}}]\right| + \Pr[bad_I^0] \\
&\leq 2 \cdot \mathbf{Adv}_{\mathsf{Ch},\mathcal{B}}^{\text{INT-CST}}(\lambda) + \left|\Pr[\mathrm{E}_{\mathcal{A}}^{1,1}] - \Pr[\mathrm{E}_{\mathcal{A}}^{1,0}]\right|.
\end{aligned}$$

We can assume without loss of generality that in game $\mathrm{E}_{\mathcal{A}}^{1,b}$ the adversary does not trigger event bad_I^b (if it does, the game stops prematurely and gives the adversary no advantage). In particular, from $\mathrm{E}_{\mathcal{A}}^{1,b}$ on we can assume that the receiving algorithm, when fed with a deviating or exceeding ciphertext fragment, either outputs error symbols or an empty string (when buffering the ciphertext and waiting for more ciphertext bits). Using the error predictability of Ch we can now predict which one of these two cases actually occurs.

To this extent we define a variant of $\mathrm{E}_{\mathcal{A}}^{1,b}$, denoted $\mathrm{E}_{\mathcal{A}}^{2,b}$, that after processing deviating or exceeding ciphertext fragments it also invokes the predictor Pred on non-genuine ciphertext fragments. More precisely, we modify $\mathcal{O}_{\mathsf{Recv}}$ by additionally computing (i) '$e \leftarrow \mathsf{Pred}(C_S, C_R, c)$' in case synchronization was lost before the adversary queries c, or (ii) '$e' \leftarrow \mathsf{Pred}(C_S, C_R, c')$ with $c' = c \,\%\, \widetilde{c}$ in case C_R deviates or exceeds C_S because of c. Then, in case (i) we compare e with the actual output m of Recv on input c, and similarly for (ii) we compare e' with the string m' accounting for the deviating or exceeding part of c. If the strings e and m, resp., e'

$\mathcal{D}(1^\lambda)^{\mathcal{O}_{\mathsf{LoR}}(\cdot,\cdot,\cdot)}$
01 $\mathsf{active} \leftarrow 0$
02 $C_S, C_R \leftarrow \varepsilon$
03 $b' \leftarrow_\$ \mathcal{A}(1^\lambda)^{\mathcal{O}_{\mathsf{LoR}}(\cdot,\cdot,\cdot),\mathcal{O}^*_{\mathsf{Recv}}(\cdot)}$
04 Return b'

If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{LoR}}(m_0, m_1, f)$:
05 $c \leftarrow \mathcal{O}_{\mathsf{LoR}}(m_0, m_1, f)$
06 $C_S \leftarrow C_S \,\|\, c$
07 Return c to $\mathcal{A}$

If $\mathcal{A}$ queries $\mathcal{O}^*_{\mathsf{Recv}}(c)$
08 If $\mathsf{active} = 1$:
09 $\quad e \leftarrow \mathsf{Pred}(C_S, C_R, c)$
10 $\quad C_R \leftarrow C_R \,\|\, c$
11 $\quad$ Return e to $\mathcal{A}$
12 Else if $C_R \,\|\, c \preccurlyeq C_S$:
13 $\quad C_R \leftarrow C_R \,\|\, c$
14 $\quad$ Return ε to $\mathcal{A}$
15 Else:
16 $\quad e \leftarrow \mathsf{Pred}(C_S, C_R, c)$
17 $\quad$ If $C_S \not\preccurlyeq C_R \,\|\, c$ or $e \neq \varepsilon$:
18 $\quad\quad \mathsf{active} \leftarrow 1$
19 $\quad C_R \leftarrow C_R \,\|\, c$
20 $\quad$ Return e to $\mathcal{A}$

Figure 3.6: *IND-CPFA adversary $\mathcal{D}$ simulates the experiment $\mathrm{E}^{2,b}_{\mathcal{A}}$ as in the proof of Theorem 1.*

and m' differ, we force termination of the game with output 0. Let bad^b_E denote the event that the outputs of Pred and Recv differ. Then $\mathrm{E}^{1,b}_{\mathcal{A}}$ and $\mathrm{E}^{2,b}_{\mathcal{A}}$ are exactly the same from $\mathcal{A}$'s point of view as long as bad^b_E does not happen, hence their difference in probability is bounded by $\Pr[bad^b_E]$.

Analogously to the previous hop we define an adversary $\mathcal{C}$ against the ERR-PRE property which runs $\mathcal{A}$, chooses a bit d uniformly at random, and simulates game $\mathrm{E}^{2,b}_{\mathcal{A}}$ by relaying $\mathcal{A}$'s queries to its oracles. Observe that, as $\mathcal{A}$ obtains no advantage if it triggers the bad event (bad^b_I) defined for the earlier hop in either of games $\mathrm{E}^{1,b}_{\mathcal{A}}$ and $\mathrm{E}^{2,b}_{\mathcal{A}}$, we can assume without loss of generality that the output of Recv to be compared with the output of the predictor will always be either an empty string or one consisting only of error symbols. Thus, any deviation of the Pred's output from the output of Recv constitutes a successful attack in the ERR-PRE experiment.

Using a similar argument to the one from the previous hop we deduce $\mathbf{Adv}^{\text{ERR-PRE}}_{\mathsf{Ch},\mathcal{C}}(\lambda) \geq \frac{1}{2}\cdot\Pr[bad^0_E] + \frac{1}{2}\cdot\Pr[bad^1_E]$, which allows us to bound the advantage of $\mathcal{A}$ in the second experiment as follows:

$$\begin{aligned}\left|\Pr[\mathrm{E}^{1,1}_{\mathcal{A}}] - \Pr[\mathrm{E}^{1,0}_{\mathcal{A}}]\right| &\leq \left|\Pr[\mathrm{E}^{1,1}_{\mathcal{A}}] - \Pr[\mathrm{E}^{2,1}_{\mathcal{A}}]\right| + \left|\Pr[\mathrm{E}^{2,1}_{\mathcal{A}}] - \Pr[\mathrm{E}^{2,0}_{\mathcal{A}}]\right| + \left|\Pr[\mathrm{E}^{2,0}_{\mathcal{A}}] - \Pr[\mathrm{E}^{1,0}_{\mathcal{A}}]\right| \\ &\leq 2 \cdot \mathbf{Adv}^{\text{ERR-PRE}}_{\mathsf{Ch},\mathcal{C}}(\lambda) + \left|\Pr[\mathrm{E}^{2,1}_{\mathcal{A}}] - \Pr[\mathrm{E}^{2,0}_{\mathcal{A}}]\right| \ .\end{aligned}$$

We finally observe that the indistinguishability game $\mathrm{E}^{2,b}_{\mathcal{A}}$ can be safely emulated using an IND-CPFA adversary $\mathcal{D}$, as shown in Figure 3.6. Here $\mathcal{D}$ is granted oracle access to $\mathcal{O}_{\mathsf{LoR}}$ as in the IND-CPFA experiment from Figure 3.2 (on page 25) and simulates oracles $\mathcal{O}_{\mathsf{LoR}}$ and $\mathcal{O}^*_{\mathsf{Recv}}$ for $\mathcal{A}$. Adversary $\mathcal{D}$ simply relays left-or-right queries to its oracle $\mathcal{O}_{\mathsf{LoR}}$, while it answers queries to $\mathcal{O}^*_{\mathsf{Recv}}$ on its own by invoking the predictor Pred and returning its output. This leads to the following relation:

$$\left|\Pr[\mathrm{E}^{2,1}_{\mathcal{A}}] - \Pr[\mathrm{E}^{2,0}_{\mathcal{A}}]\right| \leq \mathbf{Adv}^{\text{IND-CPFA}}_{\mathsf{Ch},\mathcal{D}}(\lambda) \ .$$

Combining the various bounds implied by the above sequence of game transitions yields the stated security bound. □

Error predictability vs. error simulatability. After the original publication of this work, Barwell *et al.* [BPS15a, BPS15b] introduced the notion of *error simulatability* for subtle authenticated encryption. Error simulatability is similar in spirit to, but weaker than, error predictability. Essentially, it requires the existence of an efficient simulator that produces indistinguishable output from the decryption algorithm when receiving as input invalid ciphertexts.[8]

3.5 Constructions

In this section we demonstrate the feasibility of our notions and propose a generic construction of a stream-based channel from any authenticated encryption scheme with associated data. We then prove that our construction meets strong security in terms of confidentiality and integrity. While this construction is illustrative rather than definitive, we point out that it is close to the TLS Record Protocol. Thus, our security analysis also provides a validation of the TLS channel design.

For the construction we assume an AEAD scheme that supports encryption of variable-length messages up to il bits and that generates ciphertexts of length at most $ol = 2^\nu - 1$ for a fixed $\nu \in \mathbb{N}$. The latter allows us to encode the length of each AEAD ciphertext as a bitstring of length ν. We also assume two encoding functions: $\mathsf{int2str}$ that maps integers to their bit representation and $\mathsf{str2int}$ that, vice versa, maps bitstrings to the integers they represent.

Construction 1 (Stream-based channels from AEAD)**.** *Let $il, \nu \in \mathbb{N}$ and $ol = 2^\nu - 1$, and let $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be an authenticated encryption scheme with associated data with key space $\mathcal{K}$, associated data space $\mathbb{N}$, message space $\{0,1\}^{\leq il}$, ciphertext space $\{0,1\}^{\leq ol}$, and distinguished error $\perp$. The stream-based channel $\mathsf{Ch}_{\mathsf{AEAD}} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ is defined in Figure 3.7.*

Let us discuss the algorithms of $\mathsf{Ch}_{\mathsf{AEAD}}$ in detail.

Init. The initialization algorithm generates an AEAD key K and initializes the sending and receiving states as tuples containing key K, a sequence number seqno initially set to 0, and initially empty message buffer and ciphertext buffer, respectively. The receiving state also contains a failure flag fail, initially set to 0.

Send. This algorithm buffers its input until it has collected at least il input bits. If a sufficiently long buffer is available, it invokes the AEAD encryption algorithm on input message chunks m' of length $|m'| = il$ and a running sequence number seqno as associated data.[9] The corresponding AEAD ciphertext c' is then prepended with the binary encoding of its size, i.e., with string $len = \mathsf{int2str}(|c'|)$, and the resulting bitstring $len \,\|\, c'$ is appended to the ciphertext fragment c to be output. Later in our analysis we will refer to the concatenation of an AEAD ciphertext c' prepended with its size encoding len as a 'block' $B = len \,\|\, c'$ (see line 14 in Figure 3.7). In case the algorithm was called with the flush flag set to 1, in a final step it also AEAD encrypts any remaining buffered message in the same way, in order to empty the message buffer (this message will potentially contain less than il bits). Note that the size encoding len is a bitstring

[8] More precisely, a *subtle AE* scheme is an AE scheme augmented with a 'leakage function' describing how (a specific implementation of) the decryption algorithm reacts when processing invalid ciphertexts. Thus, according to [BPS15b], the simulator's output need not be indistinguishable from the output of the decryption algorithm, but rather from the output of the leakage function. It seems plausible that Theorem 1 also holds under the weaker requirement of error simulatability.

[9] A more natural construction in the nonce-based setting would use seqno as the encryption nonce and have empty associated data input. We have chosen the current construction because of its closeness to TLS, which treats its sequence number as associated data.

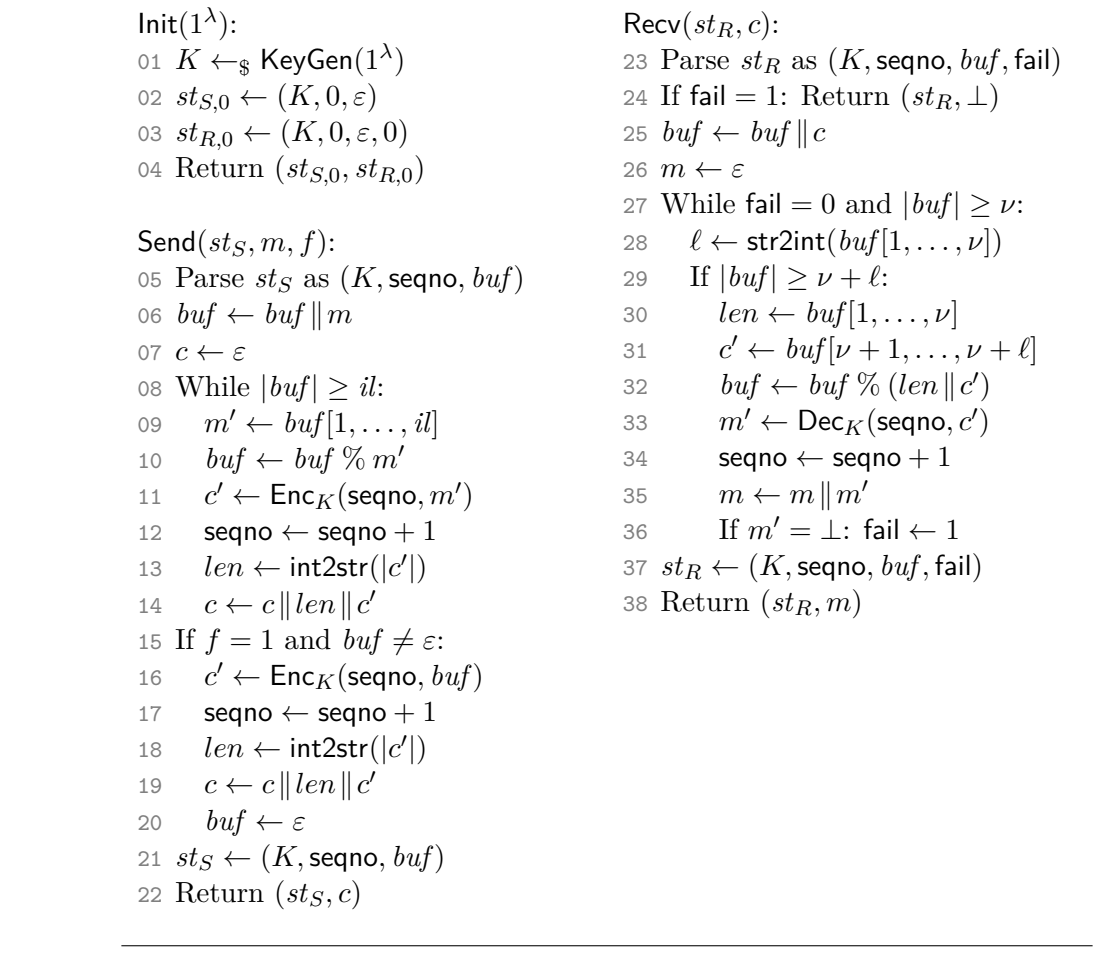


Figure 3.7: *Generic construction of a stream-based channel* $\mathsf{Ch}_{\mathsf{AEAD}} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *from an authenticated encryption scheme with associated data* $\Pi = (\mathsf{Enc}, \mathsf{Dec})$ *as specified in Construction 1. Notice that* Enc *processes up to il bits per invocation and generates ciphertexts of length at most ol. In the figure,* $|c'|$ *is the integer indicating the bit-length of ciphertext* c' *(we require* $|c'| \leq ol$*), while len is a bitstring that represents integer* $|c'|$ *(in particular,* $len \in \{0,1\}^\nu$*).*

of length ν and it is not authenticated. The algorithm returns the obtained string c and an updated state.

Recv. This algorithm outputs an error once a first error has emerged from the AEAD decryption algorithm in some previous call (this is indicated by the failure flag set to $\mathsf{fail} = 1$). Otherwise, it appends the incoming ciphertext fragment to its buffer. In case enough bits to parse the length field of ν bits were received, the algorithm computes the length ℓ of the 'next' AEAD ciphertext to be received; it then checks whether the buffer contains additional ℓ bits; if so, it extracts the complete ciphertext c' and, after incrementing the sequence number, it invokes the AEAD decryption procedure with associated data the current sequence number and ciphertext c', obtaining an output m': at this point the algorithm appends m' to the message fragment m to be output and, if decryption failed (i.e., $m' = \perp$), it sets fail to 1 and stops parsing further input, otherwise it iterates this process until there is no full block $B = len \| c'$ left in the buffer. Finally, it returns the obtained string m as well as an updated state.

Correctness. Observe that the sending algorithm generates a stream of blocks $B_1 \| B_2 \| \cdots$, $B_i = len_i \| c'_i$ where c'_i is an AEAD ciphertext generated with associated data the i-th sequence number. When processing an input fragment c the receiving algorithm first appends c to its buffer and extracts from the updated buffer the next AEAD ciphertext c'_i (the length of this ciphertext is encoded in the ν-bit prefix len of the buffer); it then AEAD decrypts the identified ciphertext c' using an increased sequence number as associated data. Now, if one delivers to Recv a prefix of the ciphertext stream that was output by Send, the receiving algorithm re-establishes the same blocks $B_i = len_i \| c'_i$ that have been processed by Send and, more importantly, it increases the sequence number by one whenever a new block is isolated: this ensures that the receiving algorithm invokes the AEAD decryption algorithm on each ciphertext c' using the same associated data that the sending algorithm had used for the AEAD encryption call that produced c'. Thus, correctness follows by virtue of the AEAD correctness.

Security. The channel $\mathsf{Ch}_{\mathsf{AEAD}}$ offers indistinguishability under chosen plaintext-fragment attacks (IND-CPFA), integrity of ciphertext streams (INT-CST), and error predictability (ERR-PRE), given that the underlying authenticated encryption scheme with associated data Π provides indistinguishability under chosen-plaintext attacks (IND-CPA) and ciphertext integrity (INT-CTXT), as defined in Section 2.3 (on page 10). By Theorem 1 we can moreover infer that $\mathsf{Ch}_{\mathsf{AEAD}}$ also provides the stronger confidentiality property of indistinguishability under chosen ciphertext-fragment attacks (IND-CCFA).

In the next three theorems we assume Π to be an authenticated encryption scheme with associated data fulfilling the properties described in Construction 1.

Theorem 2 (Confidentiality of Construction 1)**.** *If Π offers indistinguishability under chosen-plaintext attacks then $\mathsf{Ch}_{\mathsf{AEAD}}$ is indistinguishable against chosen ciphertext-fragment attacks. More precisely, for every efficient adversary $\mathcal{A}$ attacking the IND-CPFA property of Ch there exists an efficient adversary $\mathcal{B}$ against the IND-CPA property of Π such that*

$$\mathbf{Adv}^{\text{IND-CPFA}}_{\mathsf{Ch}_{\mathsf{AEAD}},\mathcal{A}}(\lambda) \leq \mathbf{Adv}^{\text{IND-CPA}}_{\Pi,\mathcal{B}}(\lambda) \ .$$

Proof. We reduce the IND-CPFA security of $\mathsf{Ch}_{\mathsf{AEAD}}$ to the IND-CPA security of Π by constructing from an efficient adversary $\mathcal{A}$ against the former property an efficient adversary $\mathcal{B}$ against the latter property. To simulate the left-or-right oracle $\mathcal{O}_{\mathsf{LoR}}$ for $\mathcal{A}$ we let $\mathcal{B}$ perform the (public) buffering and bookkeeping operations for the input messages (two buffers instead of one has to be kept for storing messages m_0 and m_1) and the sequence numbers, as defined for Send in Figure 3.7. As the buffering behavior only depends on the length of the input message to Send but not on its content, the message blocks to be AEAD encrypted are treated identically in the 'left' case ($b = 0$) and in the 'right' case ($b = 1$). This allows $\mathcal{B}$ to replace the encryption operations by encryption calls to its oracle $\mathcal{O}_{\mathsf{Enc}}$ provided by the IND-CPA game. When $\mathcal{A}$ outputs its guess b' we let $\mathcal{B}$ output the same bit and halt. Since $\mathcal{B}$ perfectly simulate the sending oracle for $\mathcal{A}$, the two adversaries have equal advantage. □

Theorem 3 (Integrity of Construction 1)**.** *If Π offers ciphertext integrity then $\mathsf{Ch}_{\mathsf{AEAD}}$ offers ciphertext-stream integrity. More precisely, for every efficient adversary $\mathcal{A}$ attacking the INT-CST property of $\mathsf{Ch}_{\mathsf{AEAD}}$ there exists an efficient adversary $\mathcal{B}$ against the INT-CTXT property of Π such that*

$$\mathbf{Adv}^{\text{INT-CST}}_{\mathsf{Ch}_{\mathsf{AEAD}},\mathcal{A}}(\lambda) \leq \mathbf{Adv}^{\text{INT-CTXT}}_{\Pi,\mathcal{B}}(\lambda) \ .$$

Proof. Recall that the receiving algorithm Recv of our channel construction processes the ciphertext stream by identifying blocks $B_1, B_2, \ldots$ with $B_i = len_i \| c'_i$ where c'_i is an AEAD ciphertext and len_i is the (fixed-length) binary encoding of its length $|c'_i|$. The message fragments output

by Recv are obtained by concatenating the AEAD decryptions m'_i of the so identified ciphertexts c'_i. In particular, Recv produces some output only if it processes at least a *full block* B_i. Relevant for this proof is that, in order to break the INT-CST property of $\mathsf{Ch}_{\mathsf{AEAD}}$, an adversary must submit to $\mathcal{O}_{\mathsf{Recv}}$ a deviating or exceeding ciphertext stream whose non-genuine part contains a full valid block $B^* = len^* \,\|\, c'^*$. More precisely, the AEAD decryption of c'^* with the current sequence number seqno as associated data yields some valid message m^*. Now, since the scheme increases the sequence number seqno before each AEAD encryption, no associated data is ever repeated. Then, under the hypothesis that $\mathcal{A}$ forges a ciphertext fragment such that a non-genuine portion of it leads to a non-empty output message, we deduce that (seqno, c'^*) is an AEAD forgery (where seqno is the current sequence number).

We now formalize the above intuition. To this end we build an adversary $\mathcal{B}$ which runs $\mathcal{A}$ internally as a black box and breaks the INT-CTXT property of Π as long as $\mathcal{A}$ is successful against the INT-CST property of $\mathsf{Ch}_{\mathsf{AEAD}}$. Adversary $\mathcal{B}$ emulates the channel construction of Figure 3.7 by forwarding AEAD encryptions to the encryption oracle $\mathcal{O}_{\mathsf{Enc}}(\cdot,\cdot)$ provided by the INT-CTXT security experiment and performing the buffering steps on its own. For this it keeps buffer strings buf_S, buf_R and a sequence number seqno, initialized to the empty strings and to zero respectively. It also keeps vectors $\mathbf{m}'$ and $\mathbf{c}'$ for bookkeeping sent AEAD messages and ciphertexts respectively, as well as a string C_R in which it registers the received stream of ciphertext fragments. It answers $\mathcal{A}$'s queries as follows.

When $\mathcal{A}$ poses a sending query (m, f), $\mathcal{B}$ appends m to the buffer buf_S, initializes an empty ciphertext c, and repeats the steps of instructions 08–14 from Figure 3.7 except for line 11 that is replaced by a call to $\mathcal{O}_{\mathsf{Enc}}$; within the loop, $\mathcal{B}$ also appends message blocks m' and ciphertext blocks c' to the vectors $\mathbf{m}'$ and $\mathbf{c}'$ respectively. If $\mathcal{A}$ had asked to flush (i.e., $f = 1$), $\mathcal{B}$ executes once more the preceding steps using as message block m' the remaining buffer. Finally it returns c to $\mathcal{A}$.

When $\mathcal{A}$ asks to decrypt a ciphertext c, $\mathcal{B}$ updates the stream C_R by appending c and checks if c is out-of-sync by comparing the components of vector $\mathbf{c}'$ (containing sent AEAD ciphertexts) with the matching blocks $B = len \,\|\, c'$ occurring within C_R. If c is genuine, $\mathcal{B}$ appends it to the buffer buf_R and then traverses the buffer to isolate the blocks B_i contained in it. If there is none, it returns an empty output to $\mathcal{A}$. Recall that for each block $B_i = len_i \,\|\, c'_i$ the AEAD ciphertext c'_i is registered in the vector $\mathbf{c}'$ and, similarly, its decryption m' is registered in $\mathbf{m}'$. Thus, $\mathcal{B}$ can for each identified ciphertext c'_i recover the decryption m'_i (as it knows the matching associated data, i.e., the i-th sequence number). After this, $\mathcal{B}$ removes the identified blocks from the buffer buf_R and concatenates all corresponding messages m'_i in the same order they appear in $\mathbf{m}'$, obtaining a string m; finally $\mathcal{B}$ gives m to $\mathcal{A}$. In case c is deviating or exceeding, $\mathcal{B}$ first performs the same procedure as above using instead of c its longest genuine prefix that contains only full blocks B_i; note that after this step the buffer buf_R will be empty (because it only contained full blocks and these are all processed). From now on $\mathcal{B}$ tries to extract a forgery either from the part of c that has not been yet processed or from $\mathcal{A}$'s subsequent receiving queries. To this end, $\mathcal{B}$ appends to buf_R the rest of c and proceeds as follows: if the buffer contains at least ν bits then it decodes $buf[1, \ldots, \nu]$ as an integer ℓ and then checks whether the remaining buffer contains ℓ more bits. If so, it outputs $c'^* = buf_R[\nu + 1, \ldots, \nu + \ell]$ together with the current sequence number increased by one as forgery. Otherwise, it gives the message fragment m obtained from the (potentially empty) in-sync part of c to $\mathcal{A}$ and waits for more receiving queries until it gets enough ciphertext bits to extract a forgery.

It is clear that $\mathcal{B}$ performs a sound simulation of the INT-CST experiment. Indeed, for answering sending queries it executes the same instructions of Send (only the AEAD encryption is replaced with an oracle call to $\mathcal{O}_{\mathsf{Enc}}$, however, the AEAD encryption takes place within the internal computations of the oracle). Note that $\mathcal{B}$ does not use its decryption oracle but can

answer all in-sync queries by looking at the message fragments that $\mathcal{A}$ requested to send. It can likewise recover the longest, genuine message-fragment underlying deviating or exceeding ciphertext fragments whose non-genuine part contains no full block $B = len \,\|\, c'$. Finally, for the non-genuine part of a deviating or exceeding ciphertext-fragment, $\mathcal{B}$ can either extract an AEAD forgery or simply answer with an empty message and wait for more ciphertext bits.

It remains to show that if $\mathcal{A}$ breaks the INT-CST property of $\mathsf{Ch}_{\mathsf{AEAD}}$ then $\mathcal{B}$ is successful in the INT-CTXT game against Π. Let c denote $\mathcal{A}$'s first out-of-sync query (i.e., either c is deviating, or it causes C_S to extend C_S and the exceeding part of c produces output), let $\widetilde{c}$ be the longest in-sync prefix of c, and let m and $\widetilde{m}$ be the message fragments that Recv would output on input c and $\widetilde{c}$ respectively in the real execution of the INT-CST experiment.

Assume first that $\mathcal{A}$ is successful with her first out-of-sync query c: we thus have $m\%[m, \widetilde{m}] \notin \mathcal{E}^*$. Suppose that the (in-sync) ciphertext stream that Recv would process up to $\widetilde{c}$ contains the first i blocks $B_1, \ldots, B_i$ that were sent. By construction $\widetilde{m} = m'_1 \,\|\, \cdots \,\|\, m'_i$ where each m'_i is the AEAD decryption of c'_i with the i-th sequence number as associated data. Then the ciphertext fragment $c \,\%\, B_1 \,\|\, \cdots \,\|\, B_i$ contains as a prefix a full block $B^* = len^* \,\|\, c'^*$ such that AEAD decrypting c'^* with associated data $\mathsf{seqno}^* = i + 1$ yields m^*.

This argument easily extends to the general case in which $\mathcal{A}$ poses multiple (out-of-sync) queries before breaking the INT-CST security of $\mathsf{Ch}_{\mathsf{AEAD}}$: $\mathcal{B}$ simply keeps buffering (and outputting an empty string as Recv would do) until it collects enough ciphertext bits to form a full block $B^* = len^* \,\|\, c'^*$ (here 'enough' means $\nu + \ell^*$ for $\ell^* = \mathsf{str2int}(len^*)$).

In both cases, once $\mathcal{B}$ obtains sufficiently many ciphertext bits to isolate a block B^*, it stops the simulation and returns as AEAD forgery the pair (seqno^*, c'^*). □

Theorem 4 (Error predictability of Construction 1)**.** *If Π offers ciphertext integrity then $\mathsf{Ch}_{\mathsf{AEAD}}$ is error predictable. More precisely, there exists an efficient error predictor Pred (that we construct in the proof) such that for every efficient adversary $\mathcal{A}$ attacking the ERR-PRE property of $\mathsf{Ch}_{\mathsf{AEAD}}$ w.r.t. Pred one can build an efficient adversary $\mathcal{B}$ against the INT-CTXT property of Π where*

$$\mathbf{Adv}^{\text{ERR-PRE}}_{\mathsf{Ch}_{\mathsf{AEAD}},\mathcal{A}}(\lambda) \leq \mathbf{Adv}^{\text{INT-CTXT}}_{\Pi,\mathcal{B}}(\lambda) \ .$$

Proof. We provide an explicit error predictor Pred for $\mathsf{Ch}_{\mathsf{AEAD}}$. Recall that a genuine stream produced by Send boils down to the concatenation of AEAD ciphertexts $c'_1, c'_2, \ldots$ (prepended with their length encoding len_i, which is necessary for isolating the ciphertexts within the stream) generated using a running sequence number as associated data. Given this (and assuming that AEAD ciphertexts cannot be forged), if the stream C_S of sent ciphertext fragments is known it is easy to forecast whether a specific ciphertext fragment submitted to Recv leads to a valid message output, to an empty output, or to an error. Indeed: (i) whenever Recv processes a full genuine block $B = len \,\|\, c'$ it will add a message component to its output; (ii) if the ciphertext fragment contains an AEAD ciphertext c' that does not match the stream C_S then c' is likely rejected by the AEAD decryption, and hence Recv will add the distinguished AEAD error $\perp$ to its output—this is true with overwhelming probability as long as Π offers integrity of ciphertexts; finally, (iii) if only a portion of a genuine block $B = len \,\|\, c'$ is available, Recv will buffer the whole input and return an empty output. Thus, a good error predictor Pred simply returns an empty string in case (i) and (iii), while it outputs $\perp$ in case (ii).

Observe that, by design, the receiving algorithm of $\mathsf{Ch}_{\mathsf{AEAD}}$ produces non-empty output only if it processes at least a full block $len \,\|\, c'$; thus, in case (iii) the error predictor behaves precisely as Recv. In both cases (i) and (ii), instead, a mismatch between the output of Recv and the output of Pred can only occur if, on input a non-genuine pair of ciphertext c'^* and associated data the current sequence number seqno, the AEAD decryption returns a valid message. Now, as the sequence number is increased after each AEAD encryption, no associated data is ever

repeated; then the pair (ad^*, c'^*) with $ad^* = \mathsf{seqno}$ is fresh and would lead to a violation of the INT-CTXT property, against the assumption. □

3.6 A Note on the TLS Record Protocol

As discussed earlier, the Transport Layer Security (TLS) Record Protocol implements a stream-based channel whose complete analysis as such lies outside of the scope of this work. However we do pause to note that our construction of a stream-based channel based on authenticated encryption with associated data from Section 3.5 is actually very close to the TLS Record Protocol when using an AEAD scheme as specified for TLS version 1.2 [DR08, Section 6.2.3.3] and in the current (as of August 17, 2016) draft for TLS version 1.3 [Res16, Section 5.2]: the Record Protocol also incorporates a sequence number which is authenticated but not sent on the wire and a length field which is sent and authenticated in TLS 1.2 (and which is sent but *not* authenticated in the draft TLS 1.3). However, the TLS Record Protocol in version 1.2 additionally includes a 2-byte version number and a 1-byte content type; these are both sent and authenticated in the associated data. Moreover, the AEAD schemes used are considered to be nonce-based, with the TLS 1.3 draft specifying how the nonce is formed and TLS 1.2 leaving the exact nonce generation to be specified by the particular cipher suite in use.

The content type field in particular allows TLS to multiplex data streams for different purposes within a single connection stream, as TLS 1.2 does for the Handshake Protocol, the Alert Protocol, the ChangeCipherSpec protocol, and the Application protocol. While our model does not capture multiplexing several message streams into one ciphertext stream, it can be augmented to do so. This brings additional complexity and is an avenue for future work.

Chapter 4

Broadcast Communication

The goals of the next three chapters is to extend existing models for cryptographic channels to the bidirectional and the broadcast settings. For this we need an appropriate communication model that enriches the traditional channel scenario, in which a sender transmits data to a receiver, to allow for interactive communication among two or more participants. In the present chapter we develop such a model.

4.1 Introduction

Security models for cryptographic channels usually assume that sender and receiver run a channel on top of a reliable network. As a consequence, in non-hostile conditions the network ensures the delivery at the receiver of the messages transmitted by the sender, according to the sending order. In this setting the communication is exclusively unidirectional, from the sender to the receiver. Here we generalize the channel scenario and assume an asynchronous network in which N participants interact with each other by exchanging messages. We focus on specific types of network that provide different delivery guarantees. The most basic network is *broadcast*, ensuring that all messages sent by any participant will, in principle, be delivered to all other participants (we write 'in principle' because we make no assumption on the delivery time). The second type of network, *FIFO broadcast*, augments plain broadcast by providing extra guarantees in terms of the relative ordering of delivery from the perspective of each sender: the FIFO ordering ensures for each pair of participants that messages sent by the one are received by the other according to the sending order. In the third network type, called *causal broadcast*, message delivery happens in a way that also preserves the causal relationships among all sending and receiving events. More precisely, a given message is delivered to any specific participant only after this participant received all messages that were sent before. Differently from FIFO broadcast, in causal broadcast the term 'before' should be understood in a *global* sense: it addresses not only the messages that a specific participant has sent but *all* messages that *all* participants sent before.

Having in mind the guarantees provided by FIFO and causal broadcast networks in an ideal world without adversaries, our goal in Chapters 5 and 6 is to enforce cryptographically that these guaranteed are also given in the presence of adversaries that control the network.

We first set some notation that will appear in the formalism later.

Notions of ordering relations. Let $(S, \leq)$ be a partial order. To indicate the *set of predecessors* and the *set of successors* of an element $s \in S$ we use the notation ${}^{\leq}s = \{s' \in S \mid s' \leq s\}$ and $s^{\leq} = \{s' \in S \mid s \leq s'\}$. Similarly we define the sets ${}^{<}s$ and $s^{<}$ of *strict predecessors* and

strict successors of s, respectively. For a subset $S' \subseteq S$ we denote with ${}^{\leq}S'$, ${}^{<}S'$, $S'^{\leq}$ and $S'^{<}$ the sets of predecessors, strict predecessors, successors, and strict successors of all elements in S', respectively. An element in $(S, \leq)$ is a *maximum* if it has no strict successor, i.e., if it is contained in $\max(S) = \{s \in S \mid \nexists s' \in S : s < s'\}$. Every strict partial order $(S, <)$ can be represented as a directed acyclic graph (DAG) with vertices in S and edges (u, v) such that $u < v$. For a binary relation $\leq$ we denote by $\leq^*$ the reflexive transitive closure of $\leq$, i.e., the smallest relation that contains $\leq$ and is reflexive and transitive. The reflexive transitive closure of a DAG is the reachability relation of the graph. For example, if u and v are vertices of a DAG and $u < v$, i.e., there is a directed edge from u to v, then $u \leq^* v$ indicates that either $u = v$ or there exists a directed path from u to v. It is a standard result that every (finite) partial order $(S, \leq)$ has a linear extension, i.e., a bijective *enumeration* $e\colon [1 \,..\, |S|] \to S$ exists such that $e(i) \leq e(j) \implies i \leq j$.

4.2 Communication Graphs

To describe how N participants interact in a broadcast fashion we introduce a combinatorial object, the *communication graph*, which describes the sending and receiving actions that participants perform and the relative order in which these actions occur.A communication graph is essentially a partial order $(V, \leq)$ augmented with a labeling function χ. The set V consists of *actions* performed by the participants. Intuitively, an action is an atomic sending or receiving operation that corresponds to a transformation of a message into a ciphertext or a ciphertext into a message. The relation $\leq$ indicates the order in which the actions of V occur, capturing a notion of logical time (which is essentially the *happened before* relation introduced in [Lam78]).[1] For any two actions $u, v \in V$ we have that $u < v$ if and only if action u *happens before* action v. The labeling function χ associates to each action a type (*sending* or *receiving*) and the participants that such action involves. More specifically, each sending action is associated with the participant that performs it (and it is considered to target all other participants), and each receiving action is associated with both the participant that performs it and with the participant the received ciphertext is assumed to originate from. We write $\chi(u) = (\mathtt{S}, i)$ to indicate that u is a sending action performed by participant i. Similarly, $\chi(v) = (\mathtt{R}, i, j)$ means that in action v participant i receives a ciphertext that is assumed to originate from participant j. Formally, if $N \in \mathbb{N}$ denotes the (fixed) overall number of participants we define $\chi\colon V \to X$ where X is the universe of possible action characteristics:

$$X = X^S \cup X^R \qquad \text{where} \qquad X^S = \{\mathtt{S}\} \times [1 \,..\, N] \qquad \text{and} \qquad X^R = \{\mathtt{R}\} \times [\![1 \,..\, N]\!] \ .$$

We call the labeling function χ the *characteristic* of the graph. Observe that this encoding does not reflect the messages and ciphertexts that might be associated with the actions (these will become relevant only in later sections of this thesis). Notice that our definitions do not allow a participant to receive ciphertexts from itself. We thus obtain the following equivalent definition of set X, where the X_i components stand for precisely the operations actively involving participant $i \in [1 \,..\, N]$:

$$X = \bigcup\nolimits_i X_i \qquad \text{where} \qquad X_i = \{(\mathtt{S}, i)\} \cup \{(\mathtt{R}, i, j) \mid j \in [1 \,..\, N] \setminus \{i\}\} \ .$$

It will prove helpful to define the following subsets of V, where we always assume $(i, j) \in [\![1 \,..\, N]\!]$: V_i^S is the subset of sending actions of participant i, V_{ij}^R is the subset of receiving actions of

[1] The irreflexive variant $<$ of the relation $\leq$ is well-known in distributed systems as *happened before* or *causality* relation since its first appearance in the celebrated work by Lamport on (distributed) logical clocks. In this thesis we refer to $\leq$ simply as the *ordering relation between actions* and, to avoid confusion, we use the term 'causality' only to address the specific ordering property that is provided, for instance, by causal broadcast networks.

participant i that are assumed to originate from participant j, V_i^R is the subset of receiving actions of participant i, $V_i = \chi^{-1}(X_i)$ is the subset of actions of participant i, $V^S = \chi^{-1}(X^S)$ is the subset of all sending actions, and $V^R = \chi^{-1}(X^R)$ is the subset of all receiving actions. Formally,

$$V_i^S = \chi^{-1}((\mathtt{S}, i)) \qquad V_{ij}^R = \chi^{-1}((\mathtt{R}, i, j)) \qquad V_i^R = \bigcup_{j \neq i} V_{ij}^R$$
$$V_i = V_i^S \cup V_i^R \qquad V^S = \bigcup_i V_i^S \qquad V^R = \bigcup_i V_i^R$$

Further, for any $V' \subseteq V$ we write $\overline{V'} = V \setminus V'$ to denote the complement of V' in V.

To define formally how the relation $\leq$ looks like we first need to specify some additional properties that we expect from broadcast communication. First of all, we assume that each participant performs at most one action at a time. This implies that the order in which any specific participant performs its actions is strictly sequential. We formalize this by requiring that $\leq$ is *locally* a total order (i.e., with respect to the actions of each individual participant), as we explain next. Let $\preccurlyeq_\ell$ (for 'local') denote the ordering relation that connects only actions of the same participant; then the actions of each specific participant form a chain with respect to $\preccurlyeq_\ell$. We also assume that any receiving action happens after the associated sending action (which must originate *remotely*, as we do not allow a participant to receive from itself), and denote the relation among actions of different participants by $\prec_r$ (for 'remote'). Finally, we assume that transmitted messages are broadcast (i.e., eventually they will be received by all participants other than the sender), and they are received at most once (messages might get lost or be still on the wire, but duplications may not happen). The latter two assumptions imply that for each receiving action $v \in V^R$ there exists a unique sending action $u \in V^S$ from which v originates. Given this, we define $\leq$ to be the coarsest poset relation simultaneously subsuming $\preccurlyeq_\ell$ and $\prec_r$. [2]

We give examples of communication graphs for $N = 3$ participants in Figure 4.1.

Definition 12 (Communication graph)**.** *Let $N \in \mathbb{N}$. A* communication graph *(with N parties) $G = (V, \leq, \chi)$ consists of a set of actions V (vertices), a partial order $\leq$ on V, and an assignment $\chi \colon V \to X$ such that there exist binary relations $\preccurlyeq_\ell$ and $\prec_r$ on V that meet the following axioms.*

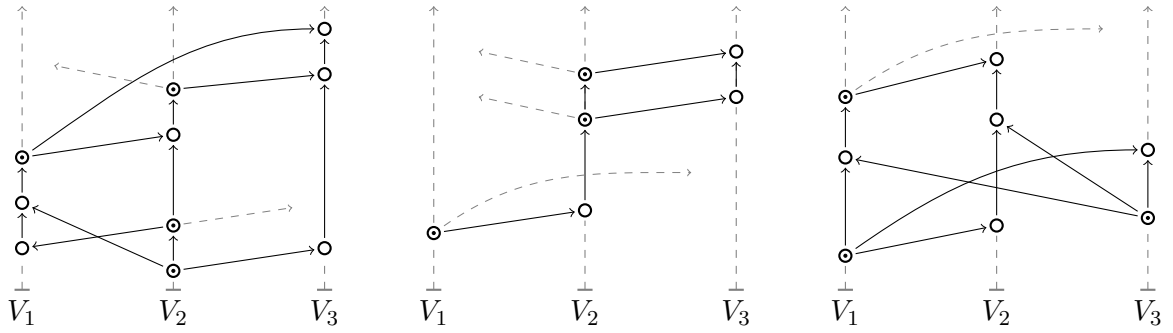
(CG1) *For any participant $i \in [1\,..\,N]$, set V_i is totally ordered by the relation $\preccurlyeq_\ell$; further, actions of distinct participants shall not be in relation, i.e., $v \in V_i \wedge v \preccurlyeq_\ell w \implies w \in V_i$. We write $\prec_\ell$ for the irreflexive variant of $\preccurlyeq_\ell$.*

(CG2) *The relation $\prec_r$ implements the characteristics of the actions and associates sending and receiving actions of remote participants; it also enforces that sending actions have at most one remote successor per other participant and that receiving actions have precisely one remote predecessor. Formally, for all $(i, j) \in [\![1\,..\,N]\!]$ we require*

$$v \in V_j^S \implies |v^{\prec_r} \cap V_i| \leq 1 \quad \textit{and} \quad |{}^{\prec_r}v| = 0$$
$$v \in V_{ij}^R \implies |{}^{\prec_r}v \cap V_j| = 1 \quad \textit{and} \quad |{}^{\prec_r}v \cap \overline{V_j}| = 0 \quad \textit{and} \quad |v^{\prec_r}| = 0 \ .$$

Note that these conditions imply that all $\prec_r$-successors of a sending action are receiving actions (if $v \in V^S$ then $v^{\prec_r} \subseteq V^R$) and that the unique $\prec_r$-predecessor of any receiving

[2] Similar abstractions of concurrent communication are common in the distributed computing literature (see [CGR11, AW04]). We deviate from the established notation and present notions that are admittedly quite heavy to grasp at a first glance. However, we believe that such a level of formalism is necessary to allow for a formal treatment of broadcast channel security. As we will see in the next chapters, the complexity of our notation will only appear in the formal proofs.



(a) *A communication graph* **(b)** *A FIFO graph* **(c)** *A causal graph*

Figure 4.1: *Illustration of 3-party communication graphs. The elements of V_1, V_2, V_3 are vertically aligned, ordered bottom-up (according to the $\prec_\ell$ relation) The vertical dashed lines symbolize per-party timelines. Sending actions are marked with ⊙ and receiving actions with ○. Non-vertical solid arrows associate sending actions to their corresponding receiving actions (i.e., indicate the $\prec_r$ relation). The graph from Figure 4.1a is neither FIFO nor causal; the graph from Figure 4.1b is FIFO but not causal, and the graph from Figure 4.1c is causal.*

action is a sending action (if $v \in V^R$ then ${}^{\prec_r}v \subseteq V^S$). We correspondingly define the origin indicator *$\omega\colon V^R \to V^S$ that maps each receiving action $v \in V^R$ to the corresponding sending action $\omega(v) \in V^S$ such that $\omega(v) \prec_r v$.*

(CG3) *The partial order relation $\leq$ is the transitive and reflexive closure $(\prec_\ell \cup \prec_r)^*$ of the combined relation $\prec_\ell \cup \prec_r$. That is, as $\prec_\ell$ and $\prec_r$ capture all considered aspects of evolving time, $\leq$ essentially establishes the global 'happened before' relation on V. We write $<$ for the irreflexive variant of $\leq$. Effectively, this axiom demands that the relation $\prec_\ell \cup \prec_r$ be 'cycle-free'.*

Observe that relations $\preccurlyeq_\ell, \prec_\ell, \prec_r$ can be uniquely recovered from $(V, \leq, \chi)$. In the course of this thesis we may hence refer to these relations without further mention.

We denote with $\mathcal{G}$ the set of all (N-party) communication graphs.

In the rest of the thesis we denote communication graphs by $G = (V, \leq, \chi)$ and illustrate them as shown in Figure 4.1a. We write $G = (\emptyset, \emptyset, \emptyset)$ in case the set of actions is empty.

4.2.1 FIFO and Causal Graphs

In this section we define the graph classes describing FIFO and causal broadcast communication. Recall that in FIFO broadcast the order of delivery is preserved with respect to all senders independently, while in causal broadcast deliveries happen in accordance to the order of all sending actions globally.

FIFO graphs. Many applications demand that the order of transmissions between any two parties always be preserved and that no sent data be missed by a receiver (in the two-party case for instance a remote shell application like SSH). We define *FIFO graphs* as the subclass of communication graphs that possess the desired 'first-in-first-out' behavior. For an example of a FIFO graph, see Figure 4.1b: here Party 2 performs two sending actions which are matched with two corresponding receiving actions at Party 3. Note that the first sending action of Party 2 is connected with the first receiving action of Party 3, as well as the second sending

is connected with the second receiving action. This is precisely due to the FIFO property. In contrast, observe that in the lower part of Figure 4.1a the transmissions caused by the first two sending actions of Party 2 'cross' on their way to Party 1. In addition, Party 3 unnoticedly misses a transmission from Party 2. While these situations are not admissible in a FIFO graph, they can occur in a communication graph.

Definition 13 (FIFO graph)**.** *We say that a communication graph $G = (V, \leq, \chi)$ is a* FIFO graph *if the following condition holds.*

(FIFO) *If a sending action of one participant reaches a second participant, then all prior sending actions of the first participant have reached the second participant before. Precisely, for all $v, w \in V^S$ and $w' \in V^R$ it holds that*

$$v \prec_\ell w \prec_r w' \qquad \Longrightarrow \qquad \exists v' \in V^R : v \prec_r v' \prec_\ell w'$$

We denote with $\mathcal{G}_{\text{FIFO}}$ the set of all (N-party) FIFO graphs.

Causal graphs. Going one step further, we define *causal graphs* as a subclass of FIFO graphs for which $(V, \leq)$ enjoys the causal order property with respect to the participant assignment established by χ. The causal order requirement captures the peculiarities of causal broadcast: It ensures that each participant performs a specific receiving action v associated to a sending action u only after having received all sending actions u' that happened before u (according to the $\leq$ relation). The upper part of Figure 4.1a is not causal in this sense as Party 3 notices the (third) sending action of Party 2 before noticing the sending action of Party 1 although, *logically*, these actions did occur in the reverse order (Party 2 might have sent as a reaction to the transmission of Party 1). Such twists of the communication history must not appear in causal graphs, as all parties at any point have a consistent view on the whole communication history up to that point. A causal graph is illustrated in Figure 4.1c.

Definition 14 (Causal graph)**.** *We say that a communication graph $G = (V, \leq, \chi)$ is a* causal graph *if the following condition holds.*

(CAUS) *If a sending action of one participant reaches a second participant, then all prior sending actions (this time 'prior' is meant in a global sense, i.e., taking into account the full set of participants) have reached the second participant before. Precisely, for all $v, w \in V^S$ and $w' \in V^R$ it holds that*

$$v < w \prec_r w' \qquad \Longrightarrow \qquad v \prec_\ell w' \ \vee \ \exists v' \in V^R : v \prec_r v' \prec_\ell w' \ .$$

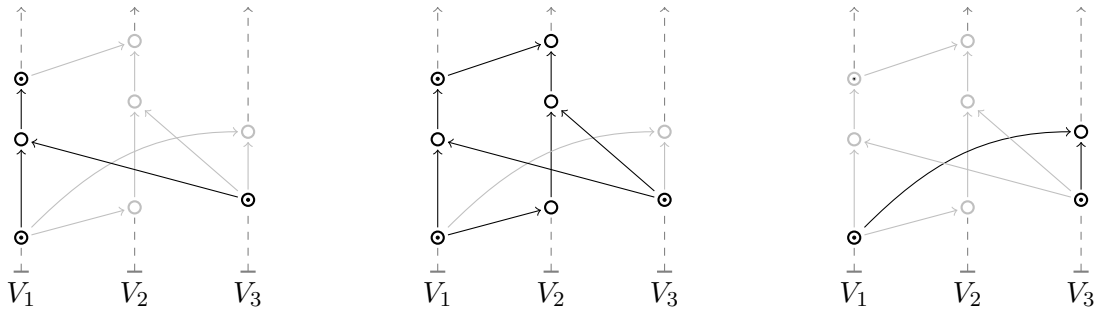
We denote with $\mathcal{G}_{\text{CAUS}}$ the set of all (N-party) causal graphs.

Note that condition (CAUS) implies (FIFO) as the $<$ relation subsumes the $\prec_\ell$ relation. With other words, every causal graph is also a FIFO graph. Since FIFO and causal graphs are special cases of communication graphs, we have $\mathcal{G}_{\text{CAUS}} \subseteq \mathcal{G}_{\text{FIFO}} \subseteq \mathcal{G}$.

4.2.2 Communication History and Graph Prefixes

Generally, participants communicating over a broadcast network can never have a *complete* view of the communication going on (for instance, how would they know whether another participant just performed a sending operation?). However, *causal* broadcast networks assert that participants, when receiving messages, can at least be sure not to have missed communication operations which logically precede their current receiving action (with other words, there is no

hole in the causal past). Thus, a participant that performs an action v is, in principle, aware of the whole 'prefix' of v in the graph, i.e., all actions ${}^{\leq}v$ that causally precede v and their relative ordering. We formalize this prefix notion as follows and refer to Figure 4.2 for an illustration.



(a) *(1)-prefix of G* **(b)** *(2)-prefix of G* **(c)** *(3)-prefix of G*

Figure 4.2: *Prefixes of a causal graph. In all three examples the graph in the background is the causal graph from Figure 4.1c, and the highlighted subgraphs are the prefixes of Party 1, Party 2, and Party 3 respectively, from left to right. Intuitively, each prefix contains all and only the actions that causally affect a given participant (i.e., that the participant should be aware of).*

Definition 15 (Prefix of a causal graph)**.** *Let $G = (V, \leq, \chi)$ be a causal graph and let $v \in V$ be an action. The* v-prefix *of G is defined as $G^{(v)} = (V', \leq', \chi')$ where $V' = {}^{\leq}v$, and $\leq'$ and χ' denote the restrictions of $\leq$ and χ, respectively, to the domain V'. Further, to define $G^{(i)}$ for $i \in [1\,..\,N]$, set $G^{(i)} = G^{(\max V_i)}$ if $V_i \neq \emptyset$ and set $G^{(i)} = (\emptyset, \emptyset, \emptyset)$ otherwise.*

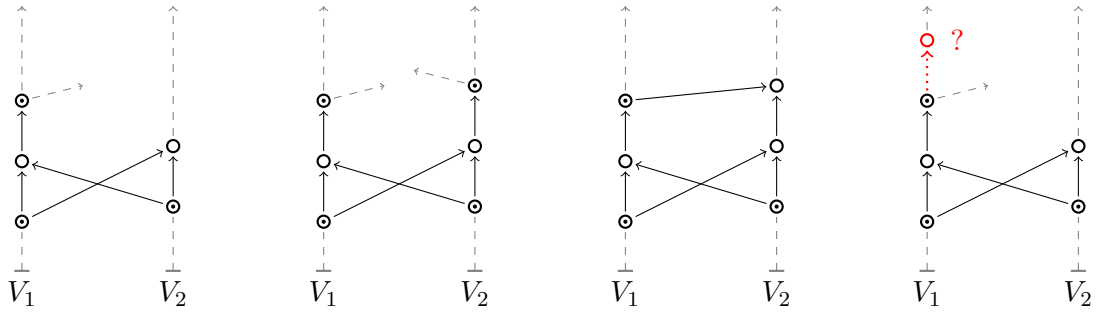
Prefixes of causal graphs are causal graphs themselves. We prove this in Lemma 3 (on page 48).

4.2.3 Modeling Dynamic Communication: Graph Addition

A communication graph represents a snapshot of some ongoing network interaction at a certain point in time. However, intuitively speaking, such interaction is a dynamic process that is continuously extended by further actions. We introduce an addition operation on communication graphs that allows to capture such modifications. For instance, we write $G' = G + (\mathtt{S}, i) + (\mathtt{R}, j, k)$ to express that communication graph G' emerges from communication graph G by first letting participant i perform a send operation and then letting participant j receive from participant k (see Figure 4.3 for an illustration). Observe that receiving operations are sometimes not permissible (in contrast to sending operations, which always are); for instance, if $G = (\emptyset, \emptyset, \emptyset)$ and $x = (\mathtt{R}, j, k)$ then operation $G + x$ is not possible, as no participant can receive if nobody had sent before. In such cases we write $G + x = \bot$. In fact, whether or not adding a receiving operation is admissible crucially depends on the type of communication graph considered (plain, FIFO, causal). In this thesis we will need addition symbols for FIFO and causal graphs. Correspondingly we define two dedicated operations

$$+ \colon \mathcal{G}_{\text{FIFO}} \times X \to \mathcal{G}_{\text{FIFO}} \cup \{\bot\} \qquad \text{and} \qquad \oplus \colon \mathcal{G}_{\text{CAUS}} \times X \to \mathcal{G}_{\text{CAUS}} \cup \{\bot\}\ ,$$

where the internal rules of $+$ are made such that if G is a FIFO graph and x is an action, then $G + x \neq \bot$ if and only if extending G by x is possible according to the FIFO graph axioms, i.e., $G + x$ is also a FIFO graph. Correspondingly, the rules of $\oplus$ are such that $G \oplus x \neq \bot$ if and only if the causal graph axioms are fulfilled by the extended graph.



(a) G **(b)** $G + (\mathtt{S}, 2)$ **(c)** $G + (\mathtt{R}, 2, 1)$ **(d)** $G + (\mathtt{R}, 1, 2)$

Figure 4.3: *Illustration of the FIFO addition (+). We start with the communication graph G from Figure 4.3a and add a sending action performed by Party 2 (Figure 4.3b), or a receiving action performed by Party 2 (Figure 4.3c). As we are in the 2-party case, by Lemma 2 the same operations would automatically be admissible if we would have used the causal addition ($\oplus$) instead. Adding a receiving action performed by Party 1 (Figure 4.3d) is not possible (Party 2 has no unmatched sendings).*

The scope of this chapter is to clarify the structure of FIFO and causal broadcast communication and introduce the communication graph framework. We will use these concepts as tools to formalize security of FIFO and causal channels in the next chapters. An intuitive understanding of the FIFO and causal properties, of the corresponding addition operations $+$ and $\oplus$, and of the concept of prefix is sufficient to move on with Chapters 5 and 6.

4.3 Technical Results

In this section we present further results concerning the structure of FIFO and causal graphs. After setting some basic properties of communication, FIFO, and causal graphs, we make the concept of graph addition formal and study how extending a graph by one action modifies the user-specific prefixes of the graph. These results should be considered as technical lemmas that will only become relevant later to prove statements regarding our channel constructions from Chapters 5 and 6. The reader may want to skip the rest of this chapter for now, and consult the specific lemmas when needed.

4.3.1 Basic Properties of Communication Graphs

The following results establish combinatorial properties of communication graphs, FIFO graphs, and causal graphs. Lemma 1 provides a numerical characterization of the $\prec_r$ relation that connects receiving actions to the sending actions they belong to. More specifically, it shows that the (FIFO) property allows associating sending and receiving actions by counting: if u is the n-th action in V_j^S and v is the m-th action in V_{ij}^R (where we count according to the $\prec_\ell$ relation), then $u \prec_r v$ if and only if $n = m$. Further, if (CAUS) holds, we prove that precisely the first $|V_{ij}^R|$-many actions in V_j^S have a successor in V_i (according to $<$, i.e., taking into account all participants). Lemma 2 shows that in the two-party setting every FIFO graph is also a causal graph.

Lemma 1 (Basic graph properties)**.** *In a communication graph $G = (V, \leq, \chi)$, let $(i, j) \in [\![1 \,..\, N]\!]$ and $s = |V_j^S|$ and $r = |V_{ij}^R|$. Then the following hold:*

(a) no participant receives more often from another participant than the latter performed sending actions: $r \leq s$.

(b) if (FIFO) holds, sending and associated receiving actions pair-up without gaps: if u is the n-th action in V_j^S (according to $\prec_\ell$) and v is the m-th action in V_{ij}^R, then $u \prec_r v$ if and only if $n = m$.

(c) if (CAUS) holds, precisely the first r actions u in V_j^S have a successor in V_i, i.e., meet condition $u \in {}^{\leq}V_i$. In particular, if $s > r$ then $V_j^S \setminus {}^{\leq}V_i \neq \emptyset$.

Proof. (a) by (CG2), every action in V_{ij}^R has precisely one $\prec_r$-predecessor in V_j^S but every action in V_j^S has at most one $\prec_r$-successor in V_{ij}^R; this implies $r \leq s$.

(b) '$\Longrightarrow$': Among all pairs u, v that fulfill the precondition but have $n \neq m$ consider w.l.o.g. the one with smallest n. If $n > m$, denote with u' the m-th action in V_j^S. By $u' \prec_\ell u \prec_r v$ and (FIFO) we know that u' has a $\prec_r$-successor in V_{ij}^R; by the minimality of u, this successor is v. Thus, v has two distinct $\prec_r$-predecessors in V_j^S, a contradiction. If $n < m$, denote with v' the n-th action in V_{ij}^R. By (CG2) there exists a unique $u' \in V_j^S$ such that $u' \prec_r v'$. The case $u' \prec_\ell u$ would contradict the minimality of u; if $u' = u$ then u would have two distinct $\prec_r$-successors in V_{ij}^R, a contradiction; and if $u \prec_\ell u' \prec_r v'$ and (FIFO) would imply the contradiction $v \prec_\ell v' \prec_\ell v$. Hence $n = m$. '$\Longleftarrow$': by (CG2) there exists a unique $u' \in V_j^S$ such that $u' \prec_r v$. By '$\Longrightarrow$', u' is the m-th action of V_j^S, i.e., $u = u' \prec_r v$.

(c) Let $u \in V_j^S$. If u is among the first r actions of V_j^S then $u \in {}^{\leq}V_i$ by (b). If u is not among the first r actions then (b) shows that there is no $v \in V_i$ such that $u \prec_r v$ (otherwise v would have two $\prec_r$-predecessors, a contradiction); by (CAUS) we then have $u \notin {}^{\leq}V_i$. □

The following results shows that, in the two-party setting, every FIFO graph is also a causal graph. This is consistent with a well-known result from distributed computing asserting that, for two communicating parties, if the FIFO property holds then so does the causal property.

Lemma 2 (Two-party case: (FIFO) = (CAUS))**.** *In the case $N = 2$, the notions of FIFO graph and causal graph are equivalent.*

Proof. Given a communication graph $G = (V, \leq, \chi)$ we need to show that (FIFO) implies (CAUS). Let thus $v, w \in V^S$ and $w' \in V^R$ such that $v < w \prec_r w'$. We either have $v \in V_1$ or $v \in V_2$. In the first case we apply the (FIFO) axiom and are done. In the second case we necessarily have $v \prec_\ell w'$ and the (CAUS) axiom is fulfilled directly. □

The following lemma establishes that prefixes of a causal graph are causal graphs themselves, that each participant is only aware of the other participants' sending actions that match his own receiving actions, and that in every prefix the maximum actions of remote participants are always sending actions.

Lemma 3 (Prefix properties)**.** *Given a causal graph $G = (V, \leq, \chi)$, let $i \in [1\,..\,N]$ and $v \in V_i$. Write $G^{(v)} = (W, \leq, \chi)$. Then the following hold:*

(a) $G^{(v)}$ is a causal graph.

(b) $G^{(v)}$ is balanced with respect to the receiving actions of participant i: $|W_{ij}^R| = |W_j^S|$ for all $j \in [1\,..\,N] \setminus \{i\}$.

(c) the last actions participant i sees of other participants are sending actions: $\max W_j \in W^S$ for all $j \in [1\,..\,N] \setminus \{i\}$.

Proof. (a) Observe that for any $u \in W$ we have ${}^{\leq}u \subseteq W$ and for any $u \in V \setminus W$ we have $u^{\leq} \cap W = \emptyset$. Given this, verifying axioms (CG1)–(CG3) and (CAUS) is immediate.

(b) Observe that by Lemma 1(c) precisely the first $|W_{ij}^R|$ sending actions of W_j^S belong to ${}^{\leq}W_i = {}^{\leq}v$.

(c) Let $u = \max W_j$. If $u \in W^S$ there is nothing to prove. If $u \in W^R$ then by (CG2) u has no $\prec_r$-successor at all, in particular not in W_i, which is a contradiction. □

4.3.2 Details of the Graph Addition

We proceed with working out the details of the addition operations for FIFO and causal graphs. Lemmas 4 and 5 provide a characterizations of FIFO and causal graphs that clarify under which conditions these graph types may be extended via their respective addition operations.

Lemma 4 (FIFO connectivity)**.** *In a FIFO graph $G = (V, \leq, \chi)$, let $(i, j) \in [\![1 \,..\, N]\!]$. Consider a sending action $u \in V_j^S$ and a receiving action $v \in V_{ij}^R$, and let $V_i' = {}^{<}v \cap V_i = {}^{\prec_\ell}v$ denote the 'past' of v within V_i. Then u and v are in $\prec_r$ relation if and only if all sending predecessors of u in V_j are 'received' in V_i', but u itself is not. Formally, $u \prec_r v$ if and only if ${}^{\prec_\ell}u \cap V^S \subseteq {}^{\prec_r}V_i'$ and $u \notin {}^{\prec_r}V_i'$.*

Proof. '$\Longrightarrow$': For all $w \in {}^{\prec_\ell}u \cap V^S$ we have $w \prec_\ell u \prec_r v$ and hence $w \in {}^{\prec_r}V_i'$ by (FIFO). Further, if we had $u \in {}^{\prec_r}V_i'$, then u would have two distinct $\prec_r$-successors in V_i, a contradiction. Hence $u \notin {}^{\prec_r}V_i'$. '$\Longleftarrow$': By (CG2) there exists $u' \in V_j^S$ with $u' \prec_r v$; we need to show $u' = u$. Indeed, if $u' \prec_\ell u$ then by assumption we have $u' \in {}^{\prec_r}V_i'$; as we also assume $u \notin {}^{\prec_r}V_i'$, this shows $u' \neq u$.

On the other hand, if $u \prec_\ell u'$ then by (FIFO) there exists $v' \in V_i$ such that $u \prec_r v' \prec_\ell v$, contradicting $u \notin {}^{\prec_r}V_i'$. Thus $u' = u$. □

Lemma 5 (Causal connectivity)**.** *In a causal graph $G = (V, \leq, \chi)$, let $(i, j) \in [\![1 \,..\, N]\!]$. Consider a sending action $u \in V_j^S$ and a receiving action $v \in V_{ij}^R$, and let $V_i' = {}^{<}v \cap V_i = {}^{\prec_\ell}v$ denote the 'past' of v within V_i. Then u and v are in $\prec_r$ relation if and only if all sending predecessors of u are 'known' in V_i', but u itself is not. Formally, $u \prec_r v$ if and only if ${}^{<}u \cap V^S \subseteq {}^{\leq}V_i'$ and $u \notin {}^{\leq}V_i'$.*

Proof. '$\Longrightarrow$': For all $w \in {}^{<}u \cap V^S$ we have $w < u \prec_r v$ and hence $w \in {}^{\leq}V_i'$ by (CAUS). Further, if we had $u \in {}^{\leq}V_i'$, then by (CAUS) there would exist $v' \in V_i$ such that $u \prec_r v' \prec_\ell v$, i.e., u would have two distinct $\prec_r$-successors in V_i, a contradiction. Hence $u \notin {}^{\leq}V_i'$. '$\Longleftarrow$': By (CG2) there exists $u' \in V_j^S$ with $u' \prec_r v$; we need to show $u' = u$. Indeed, if $u' \prec_\ell u$ then by assumption we have $u' \in {}^{\leq}V_i'$ and by (CAUS) there exists $v' \in V_i'$ such that $u' \prec_r v'$. As necessarily $v' \prec_\ell v$ holds, u' would have two distinct $\prec_r$-successors in V_i, a contradiction. On the other hand, if $u \prec_\ell u'$ then by (CAUS) there exists $v' \in V_i$ such that $u \prec_r v' \prec_\ell v$, contradicting $u \notin {}^{\leq}V_i'$. Thus $u' = u$. □

We are now able to fully define the addition operations on graphs and formalize the details of the $+$ and $\oplus$ operations, i.e., the mappings $+ \colon \mathcal{G}_{\text{FIFO}} \times X \to \mathcal{G}_{\text{FIFO}} \cup \{\bot\}$ and $\oplus \colon \mathcal{G}_{\text{CAUS}} \times X \to \mathcal{G}_{\text{CAUS}} \cup \{\bot\}$. In summary, adding a sending operation is always possible, and the graph is augmented by extending the local ($\prec_\ell$) relation to include the edges connecting all actions of the participant performing the action with this newly added action (below, all actions in V_i are connected with sending action v^*). Adding a receiving operation, instead, is only possible if the (FIFO) axiom, respectively, the (CAUS) axiom, are fulfilled and, in this case, the graph is augmented by extending the local ($\prec_\ell$) relation as above, as well as the remote ($\prec_r$) relation by adding the edge connecting the newly added (receiving) action with its matching sending action (below, sending action u is connected with receiving action v^*).

Definition 16 (The $+$ operation, or FIFO addition)**.** *Let $G = (V, \leq, \chi)$ be a FIFO graph and let $x \in X$. Let either $x = (\mathtt{S}, i)$ with $i \in [1 \,..\, N]$ or $x = (\mathtt{R}, i, j)$ with $(i, j) \in [\![1 \,..\, N]\!]$. Let*

$V^* = V \cup \{v^*\}$ *where* $v^* \notin V$ *is an auxiliary action and let* $\chi^*\colon V^* \to X$ *be an assignment that coincides with* χ *on* V *and additionally maps* $v^* \mapsto x$. *Let* ${\prec_\ell}' = {\prec_\ell} \cup (V_i \times \{v^*\})$. *To define operations*

$$G + (\mathtt{S}, i) \qquad \text{and} \qquad G + (\mathtt{R}, i, j)$$

we distinguish the following three cases:

- *if* $x = (\mathtt{S}, i)$ *we set* $G + x = (V^*, \leq^*, \chi^*)$ *where* $\leq^* = ({\prec_\ell}' \cup {\prec_r}')^*$ *with* ${\prec_r}' = {\prec_r}$;
- *if* $x = (\mathtt{R}, i, j)$ *and there exists* $u \in V_j^S \setminus {\prec_r} V_i$ *such that* ${\prec_\ell} u \cap V^S \subseteq {\prec_r} V_i$ *we set* $G + x = (V^*, \leq^*, \chi^*)$ *where* ${\prec_r}' = {\prec_r} \cup \{(u, v^*)\}$;
- *otherwise, we set* $G + x = \bot$.

Definition 17 (The $\oplus$ operation, or causal addition)**.** *Let* $G = (V, \leq, \chi)$ *be a causal graph and let* $x \in X$. *Let either* $x = (\mathtt{S}, i)$ *with* $i \in [1\,..\,N]$ *or* $x = (\mathtt{R}, i, j)$ *with* $(i, j) \in [\![1\,..\,N]\!]$. *Let* $V^* = V \cup \{v^*\}$ *where* $v^* \notin V$ *is an auxiliary action and let* $\chi^*\colon V^* \to X$ *be an assignment that coincides with* χ *on* V *and additionally maps* $v^* \mapsto x$. *Let* ${\prec_\ell}' = {\prec_\ell} \cup (V_i \times \{v^*\})$. *To define operations*

$$G \oplus (\mathtt{S}, i) \qquad \text{and} \qquad G \oplus (\mathtt{R}, i, j)$$

we distinguish the following three cases:

- *if* $x = (\mathtt{S}, i)$ *we set* $G \oplus x = (V^*, \leq^*, \chi^*)$ *where* $\leq^* = ({\prec_\ell}' \cup {\prec_r}')^*$ *and* ${\prec_r}' = {\prec_r}$;
- *if* $x = (\mathtt{R}, i, j)$ *and there exists* $u \in V_j^S \setminus {\leq} V_i$ *such that* ${\prec} u \cap V^S \subseteq {\leq} V_i$ *we set* $G \oplus x = (V^*, \leq^*, \chi^*)$ *where* $\leq^* = ({\prec_\ell}' \cup {\prec_r}')^*$ *and* ${\prec_r}' = {\prec_r} \cup \{(u, v^*)\}$;
- *otherwise, we set* $G \oplus x = \bot$.

To establish the well-definedness of Definition 16 we need to show that if $G \in \mathcal{G}_{\text{FIFO}}$ then $G' = G + x \neq \bot$ implies $G' \in \mathcal{G}_{\text{FIFO}}$. This clearly holds in the $x = (\mathtt{S}, i)$ case (performing a sending can never violate the (FIFO) property). The $x = (\mathtt{R}, i, j)$ case is covered by Lemma 4 (where V_i' takes the role of V_i). Similarly, we need to show that if $G \in \mathcal{G}_{\text{CAUS}}$ then $G' = G \oplus x \neq \bot$ implies $G' \in \mathcal{G}_{\text{CAUS}}$ for Definition 17 to be well-defined. The $x = (\mathtt{S}, i)$ case is immediate (sending cannot violate the (CAUS) property either), while the $x = (\mathtt{R}, i, j)$ case is covered by Lemma 5.

We proceed with studying important relations between the operations $+$ and $\oplus$ that will be needed in the next chapters. First of all, we show that among the two, $\oplus$ is the strictly stronger operation, i.e., whenever adding an operation with $\oplus$ is admissible, then adding it with $+$ is also admissible.

Lemma 6 ($\oplus$ implies $+$)**.** *Let* $G = (V, \leq, \chi)$ *be a causal graph and let* $x \in X$ *be an action such that* $G \oplus x \neq \bot$. *Then* $G + x = G \oplus x$.

Proof. We first prove that $G + x \neq \bot$ and, in a second step, we will show that $G + x = G \oplus x$. If $x = (\mathtt{S}, i)$ there is nothing to prove because as adding a sending action is always possible. Let $x = (\mathtt{R}, i, j)$. By the hypothesis that $G \oplus (\mathtt{R}, i, j) \neq \bot$ we know from Definition 17 that there exists an action $u \in V_j^S \setminus {\leq} V_i$ such that ${\prec} u \cap V^S \subseteq {\leq} V_i$. By definition of the $\prec_r$ relation it follows that ${\prec_r} V_i$ is a subset of ${\leq} V_i$ (this holds for communication graphs in general and for FIFO and causal graphs in particular), and hence the action u belongs to the set $V_j^S \setminus {\prec_r} V_i$. Since G is a causal graph, it follows from Lemma 7 that ${\leq} V_i \cap V^S \subseteq V_i \cup {\prec_r} V_i$. The latter relation immediately implies that ${\prec_\ell} u \cap V^S \subseteq {\prec} u \cap V^S \subseteq {\prec_r} V_i$ and concludes the first part of the proof. For proving that $G + x$ and $G \oplus x$ are the same graph note that they are obtained from the same graph $G = (V, \leq, \chi)$ by adding, in both cases, a node v^* to V and the same pairs to the relation $\leq$ (this is evident by inspecting the details of Definitions 16 and 17). $\square$

The following Lemma (used in the proof of Lemma 6) establishes that, in a causal graph, the sets of sending actions that 'eventually reach' a given participant (i.e., the set ${}^{\leq}V_i$ if i is the participant in question) consists of the actions of that participant (i.e., V_i) and those that 'directly reach' that participant (i.e., ${}^{\prec_r}V_i$). This basically means that if there is a sending action v that causally precede (or 'happens before') some action in V_i, then Party i should be aware of this sending: either Party i performed action v on its own, or it performed the corresponding receiving actions w such that $v \prec_r w$. This is a direct consequence of Axiom (CAUS).

Lemma 7. *Let $G = (V, \leq, \chi)$ be an N-party causal graph. Then for all $i \in [1\,..\,N]$ it holds ${}^{\leq}V_i \cap V^S \subseteq V_i \cup {}^{\prec_r}V_i$.*

Proof. Let $v \in {}^{\leq}V_i \cap V^S$. If $v \in V_i$ the statement is trivially fulfilled. Assume $v \notin V_i$. Then there exist actions $w \in V^S$ and $w' \in V_i^R$ such that $v \leq w \prec_r w'$. If $v = w$ we are done. Otherwise, it follows by Axiom (CAUS) that either $v \prec_\ell w'$ which contradicts the assumption $v \notin V_i$, or there exists $v' \in V^R : v \prec_r v' \prec_\ell w'$, which implies that $v \in {}^{\prec_r}V_i$ and concludes the proof. □

4.3.3 Incrementing a graph's prefix

In Lemma 8 we study how the augmentation of a causal graph using the $\oplus$ operation is perceived by the individual participants (recall that the (i)-prefix of a causal graph G represents only the part of G that Party i is aware of, so while some increments will be reflected in the updated $G^{(i)}$, others will not). The latter result plays a key role in proving correctness and security of our channel construction from Chapter 6.

We first introduce the following notion of 'projected characteristic', which describes the actions performed (locally) by a specific participant. Concretely, the projected characteristic of Party i is simply the string obtained by concatenating the characteristics of all actions in V_i, according to the order in which these actions occur. Then, we use this notion directly in Lemma 8.

Definition 18 (Projected characteristic)**.** *Let $G = (V, \leq, \chi)$ be a communication graph, let $i \in [1\,..\,N]$, and let $V_i = \{v_{i,1}, \ldots, v_{i,n_i}\}$ such that $v_{i,1} \prec_\ell \ldots \prec_\ell v_{i,n_i}$ and $n_i = |V_i|$. The projection $\chi_i \colon \mathcal{G} \to X^*$ is defined such that $\chi_i(G) = \chi(v_{i,1}) \ldots \chi(v_{i,n_i})$.*

We have seen under which conditions it is possible to add an action to a causal graph and how the resulting graph looks like. The following lemma describes how each prefix of the graph is modified by this addition. With other words, it says which participants are affected by the newly added action and how their view of the ongoing communication changes. Briefly, it formalizes the following intuitive facts: (i) the added action is only visible to the participant that performs it, (ii) a sending action increments the participant's prefix by one action, namely the action itself, and (iii) a receiving action always increments the participant's prefix by two actions, i.e., the action itself and the matching sending action of the alleged sender, and if the alleged sender performed some receiving operations these are also added to the graph prefix.

Lemma 8 (Incrementing prefix)**.** *Let G be a causal graph. Let $x \in X$ be an action such that also $G' = G \oplus x$ is a causal graph. Then the following statements hold.*

(a) If $x = (\mathtt{R}, i, j)$ for some $(i, j) \in [\![1\,..\,N]\!]$, there exists a string $I_j = \iota^1 \parallel \ldots \parallel \iota^t \in [1\,..\,N]^*$ such that

$$\chi_j(G'^{(i)}) = \chi_j(G^{(i)}) \parallel (\mathtt{R}, j, \iota^1) \parallel \ldots \parallel (\mathtt{R}, j, \iota^t) \parallel (\mathtt{S}, j) \ .$$

(b) For all $i \in [1\,..\,N]$ we have

$$G'^{(i)} = \begin{cases} G^{(i)} & \text{if } x \notin X_i \\ G^{(i)} \oplus (\mathtt{S}, i) & \text{if } x = (\mathtt{S}, i) \\ G^{(i)} \oplus (\mathtt{R}, j, \iota^1) \oplus \ldots \oplus (\mathtt{R}, j, \iota^t) \oplus (\mathtt{S}, j) \oplus (\mathtt{R}, i, j) & \text{if } x = (\mathtt{R}, i, j) \end{cases},$$

where in the third line we use the assignment of $\iota^1, \ldots, \iota^t$ from (a).

Proof. Cases $x \notin X_i$ and $x = (\mathtt{S}, i)$ of (b) are clear by Definition 17. We hence assume $x = (\mathtt{R}, i, j)$ and prove (a) and the remaining part of (b) together. Write $G = (V, \leq, \chi)$ and $G' = (V', \leq, \chi)$ and $G^{(i)} = (W, \leq, \chi)$ and $G'^{(i)} = (W', \leq, \chi)$. Let $v = \max V_i'$ be the action added by the $\oplus$ operation, and let $w = \max W_j$ and $w' = \max W_j'$. Consider set $\Delta = (W' \setminus \{v\}) \setminus W$ by which $G^{(i)}$ is extended by the $\oplus$ operation, not counting v. From Lemma 3(b) we know that precisely one action in Δ is a sending action (otherwise, corresponding receiving actions would be missing in V_i'); by Lemma 3(c), this action is w'. As all remaining actions of Δ are in W'^R, again using Lemma 3(c), we deduce $\Delta \subseteq W_j'$. More precisely, we can write $\Delta = \{v_1, \ldots, v_t, w'\}$ such that $w \prec_\ell v_1 \prec_\ell \ldots \prec_\ell v_t \prec_\ell w'$. This shows (a).

It remains to show that the sequence of $\oplus$ operations in (b) is admissible. To see this, let $u_k = \omega(v_k)$ for all k and observe that the admissibility condition from Lemma 5 is in fact a characterization, and that it depends only on the predecessors of u_k and v_k. That is, observation $u_1, \ldots, u_t \in W$ and an inductive argument show that for $1 \leq k \leq t$ the corresponding '$\oplus(\mathtt{R}, j, \iota^k)$' operation will not fail. □

Chapter 5

FIFO Channels

In this chapter we develop functionality and security for FIFO channels. We then show how to construct secure FIFO channels from simpler cryptographic building blocks.

5.1 Introduction

Cryptographic analyses of real-world channel protocols like the TLS record protocol [DR08] and SSH Binary Packet Protocol [YL06] usually model secure channels as stateful authenticated encryption primitives [BKN02, JKSS12, BSWW13]. We note, however, that although stateful encryption was introduced for analyzing (and improving) the bidirectional SSH protocol, the notion is rather an approximation of a unidirectional channel.

A bidirectional channel can be constructed by running two instances of a unidirectional channel in reverse directions. That is, messages from Alice to Bob are sent through the one channel, and messages from Bob to Alice through the other. We call this construction the *canonic composition* of two unidirectional channels. Bidirectional channel protocols that follow this paradigm include TLS and SSH. For instance, the TLS handshake establishes a total of four keys to be used by the TLS record layer: two keys to protect against message tampering and eavesdropping in the one direction, and two keys to protect the reverse direction. When using this approach it might seem plausible that any desired level of confidentiality and integrity of the composed channel could be achieved by choosing sufficiently confidential and integrous unidirectional channels. For instance, for obtaining confidentiality against active adversaries in the bidirectional case, it might seem that requiring IND-CCA security (Chapter 2) of the unidirectional channels would suffice. This intuition turns out to be wrong in general, as the following example illustrates.

Canonic composition of unidirectional channels and its (in)security. Consider an instant messaging service that allows registered users, after authenticating with a password, to chat with any other user of the service. Alice and Bob engage in a conversation. Since Alice cares about privacy, she insists on running the service over a bidirectional cryptographic channel that offers confidentiality against active attacks. If Alice and Bob follow the canonic composition paradigm and communicate using two independent IND-CCA-secure unidirectional channels, do they achieve the desired level of security? They do not. Indeed, assume the encryption system is such that the adversary is able to inject ciphertexts that decrypt to messages of her choice.[1] Under this condition, here is how the adversary proceeds (see Figure 5.1 for an illustration). It

[1] This assumption does not contradict a pure confidentiality notion: IND-CCA security only requires that the outputs of the decryption algorithm in case of an active attack be *independent* of the encrypted messages.

delivers in the $B \to A$ direction a ciphertext that Alice decrypts to 'please authenticate'; Alice answers by sending her password over the $A \to B$ channel; as Alice's message comes unexpected and Bob cannot make sense out of it, he puts the password on public display; the adversary learns it from there.

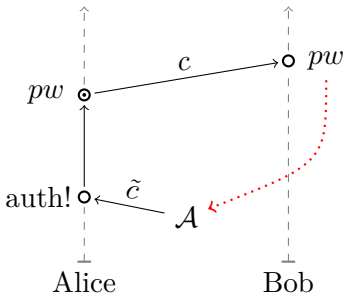


Figure 5.1: *A confidentiality attack against the canonic composition of two IND-CCA-secure unidirectional channels. In the figure time evolves bottom-up (dashed lines). Recall from Chapter 4 that nodes marked as ⊙ represent sending actions and nodes marked as ○ represent receiving actions.*

Intuitively, a bidirectional channel with *confidentiality against active adversaries* should prevent this attack from working (more precisely: it does not have to identify and report the attack but ensure that any information that Bob recovers under attack and potentially makes public be independent of what Alice sent). Evidently, the canonic composition falls short in providing this kind of protection. As the described attack involves tampering with ciphertexts, one could come to the conclusion that requiring the unidirectional channels to provide integrity in addition to confidentiality would solve the problem. Is this change sufficient? Is it necessary? Does this approach resolve *all* possible issues arising in bidirectional communication? We do not question that adding integrity protection to a symmetric channel is a good idea in general. However, making integrity a necessary part of the model also obstructs the view on the core of its security properties. We believe that rigorously answering the above questions is impossible without first defining/understanding *what security actually means* in the bidirectional case.

Having in mind the goal of understanding secure communication between two interacting participants, we go one step further and envision a multi-party scenario where an arbitrary number of users interact with all the other users. For this we propose the notion of a *broadcast channel*, allowing a group of participants to exchange messages securely in a broadcast fashion: all participants may transmit, and all transmissions target the whole group (as opposed to individuals). Such a setting is standard, for example, in Internet chat rooms, but also automated communication systems, e.g., interconnected bank computers, rely on such infrastructure. Clearly, the notion of bidirectional channel is a special case of broadcast channel that supports two participants.

In the next two chapters we exclusively consider broadcast channels, i.e., sending operations always target all participants (except the sender itself).

5.2 Syntax and Functionality

A FIFO channel should allow two or more parties to communicate securely in a broadcast fashion. As a natural generalization of channels that run on top of a reliable network like TCP/IP, we expect the following delivery guarantee for the users of a FIFO channel: for any pair of participants, the messages sent by one of the participants are received by the other participant in the same order they were sent. Thus, we assume that FIFO channels run on top of a FIFO

broadcast network (Chapter 4). Going one step ahead, we require that FIFO channels preserve the delivery guarantees of the underlying network as long as no active adversary tampers with the transmission.

We introduce the syntactical model for FIFO channels in the next definition.

Definition 19 (Syntax of FIFO channels)**.** *A* FIFO channel *with associated data space $\mathcal{AD}$, message space $\mathcal{M}$, ciphertext space $\mathcal{C}$, and state space $\mathcal{S}$ is a tuple* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *of efficient algorithms as follows:*

- Init*. The initialization algorithm takes a security parameter 1^λ and an integer N, and outputs initial states $st_1, \ldots, st_N \in \mathcal{S}$. We write $(st_1, \ldots, st_N) \leftarrow_\$ \mathsf{Init}(1^\lambda, N)$.*
- Send*. The sending algorithm takes as input a state $st \in \mathcal{S}$, associated data $ad \in \mathcal{AD}$, and a message $m \in \mathcal{M}$, and outputs a state $st' \in \mathcal{S}$ and a ciphertext $c \in \mathcal{C}$ or $c = \perp$. We write $(st', c) \leftarrow_\$ \mathsf{Send}(st, ad, m)$.*
- Recv*. The receiving algorithm takes a state $st \in \mathcal{S}$, an origin indicator $j \in [1\,..\,N]$, associated data $ad \in \mathcal{AD}$, and a ciphertext $c \in \mathcal{C}$, and outputs a state $st' \in \mathcal{S}$ and a message $m \in \mathcal{M}$ or $m = \perp$. We write $(st', m) \leftarrow_\$ \mathsf{Recv}(st, j, ad, c)$. In case $m = \perp$ we say that the algorithm rejects; otherwise, we say that it accepts.*

We assume that upon returning a decryption error $m = \perp$ the algorithm Recv *enters an error state and sets $st' = \perp$; we further require that, on input 'state' $\perp$, both* Send *and* Recv *output $(\perp, \perp)$.*

If required, both the send and the receive operations can also take associated data [Rog02] that is assumed to match on both sides. Note that our syntax also allows Send to reject. For simplicity, if an error occurs (this is notified by outputting the distinguished symbol $\perp$) we instruct the channel algorithms to reject all subsequent invocations by setting the state variable to $st = \perp$. This reflects the reasonable behavior of (cryptographic) applications which, upon being notified of an error, erase all current state information and refuse to process all further input.[2] In principle, one could extend the model and allow Send and Recv to output distinguishable errors from an error space $\mathcal{E}$ (such that $|\mathcal{E}| > 1$ similarly to the syntax of stream-based channels from Chapter 3).

Correctness for FIFO channels mirrors the guarantees induced by the assumed underlying network (FIFO broadcast, see Chapter 4). Intuitively, if participants schedule their sending and receiving actions according to a FIFO order, then they can recover all messages transmitted using Send by feeding the corresponding ciphertexts to Recv. We make this condition precise in the following definition, which heavily relies on the formalism developed in Chapter 4. More explicitly, we translate that 'users schedule their sending and receiving actions according to a FIFO order' by modeling honest communication patterns as FIFO graphs. Given such a graph, we run sequentially Send and Recv for each of its sending and receiving actions, respectively, according to an arbitrary enumeration of the graph's actions—this ensures that the scheduled communication does satisfy the FIFO property— in a way that every ciphertext $c[v]$ output by Send for a given action v is processed by Recv on the matching receiving action w, i.e., such that $v \prec_r w$ (for the meaning of the symbol $\prec_r$ see Chapter 4). Then, we define correctness in the natural way: messages input to Send and messages output by Recv on matching actions coincide.

[2] We could also have opted for a more general syntax at the cost of introducing unnecessary complexity to the integrity experiments. Instead, we opted for excluding syntactically from our treatment all channels that keep processing their input after the first errors.

Definition 20 (Correctness of FIFO channels)**.** *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a FIFO channel with associated data space* $\mathcal{AD}$*, message space* $\mathcal{M}$*, ciphertext space* $\mathcal{C}$*, and state space* $\mathcal{S}$*. Let* $G = (V, \leq, \chi)$ *be an* N*-party communication graph and for* $n = |V|$ *let* $e\colon [1\,..\,n] \to V$ *be an enumeration of* $(V, \leq)$*. Consider arbitrary assignments* $\alpha\colon V^S \to \mathcal{AD}$ *and* $\mu\colon V^S \to \mathcal{M}$ *of sending actions to associated data and messages. Denote by* $c[\,]$ *and* $m[\,]$ *associative arrays that map* $V^S \to \mathcal{C}$ *and* $V^R \to \mathcal{M}$*, respectively. Recall from Definition 12 (on page 43) that* $\omega\colon V^R \to V^S$ *denotes the originator mapping. Consider the following procedure:*

---

01 Initialize states $(st_1, \ldots, st_N) \leftarrow_\$ \mathsf{Init}(1^\lambda, N)$
02 Process the actions $v \in V$ in order $v_1 = e(1), \ldots, v_n = e(n)$ according to the rules:
03 – if $\chi(v) = (\mathtt{S}, i)$ then set $(st_i, c[v]) \leftarrow_\$ \mathsf{Send}(st_i, \alpha(v), \mu(v))$
04 – if $\chi(v) = (\mathtt{R}, i, j)$ then set $(st_i, m[v]) \leftarrow_\$ \mathsf{Recv}(st_i, j, \alpha(\omega(v)), c[\omega(v)])$

We say that a FIFO channel Ch *is correct if for every FIFO graph* $G \in \mathcal{G}_{\text{FIFO}}$*, for all* e, α, μ*, and for all choices of the randomness for* Init*,* Send*, and* Recv*, the* Recv *algorithm in the procedure above correctly recovers all sent messages, i.e., for all* $v \in V^R$ *we have* $m[v] = \mu(\omega(v))$*.*

5.3 Defining Security for FIFO Channels

In this section we describe and formalize the expected security properties for FIFO channels. Intuitively, a FIFO channel should add cryptographic protection to a FIFO broadcast network. We model this by extending the security notions for stateful authenticated encryption (Chapter 2) to a setting where multiple parties interact with each other. Throughout this section we develop indistinguishability and integrity games in which an adversary $\mathcal{A}$ interacts with the Send and Recv algorithms through oracles. Following the approach of Bellare *et al.* [BKN02], in the indistinguishability game we let the adversary query a left-or-right and a receiving oracle on arbitrary chosen message pairs and chosen ciphertexts respectively. Clearly, some of the queries to the receiving oracle would lead to trivial wins, and we need to instruct the oracle to suppress the output of Recv in this case. Thus, the challenge here is to determine which receiving queries shall be considered 'passive' (a.k.a. 'in-sync'). Similarly, in the integrity game we need to translate what it means to violate plaintext integrity or ciphertext integrity in the FIFO setting.

From stateful encryption (a.k.a. unidirectional channels) to FIFO channels. We formalize security for FIFO channels by first identifying what an active attack is. Recall that in the case of stateless encryption an adversary is active if it tampers with the sent ciphertexts. In stateful encryption, an adversary is active also if it tampers with the order of the sent ciphertexts; indeed, delivering ciphertexts in a different order than the sending order violates the properties of the network underlying a stateful encryption scheme and, thus, represents an active measure of the adversary. The same holds for FIFO channel: an adversary has to be considered active not only if it modifies a ciphertext but also if it submits sending and receiving queries that violate the FIFO ordering properties. This observation is the leading principle behind our security notions for FIFO channels, as we explain next.

In Figure 5.2 we specify the experiments of *indistinguishability under chosen-plaintext attacks* (F-IND-CPA) and under *chosen-ciphertext attacks* (F-IND-CCA) for FIFO channels. Given the experiments, we define security as follows.

Definition 21 (Indistinguishability for FIFO channels)**.** *For* $\text{ATK} \in \{\text{CPA}, \text{CCA}\}$ *we say that a FIFO channel* Ch *offers* ATK-indistinguishability *if for all efficient adversaries* $\mathcal{A}$ *and all*

polynomials $N = N(\lambda)$ the following advantage function is negligible,

$$\mathbf{Adv}^{\text{F-IND-ATK}}_{\mathsf{Ch},N,\mathcal{A}}(\lambda) := \left|\Pr\left[\text{Expt}^{\text{F-IND-ATK},1}_{\mathsf{Ch},N,\mathcal{A}}(1^\lambda) = 1\right] - \Pr\left[\text{Expt}^{\text{F-IND-ATK},0}_{\mathsf{Ch},N,\mathcal{A}}(1^\lambda) = 1\right]\right| \ .$$

We abbreviate indistinguishability under chosen-plaintext attacks (CPA-indistinguishability) and indistinguishability under a chosen-ciphertext attacks (CCA-indistinguishability) for FIFO channels by writing F-IND-CPA and F-IND-CCA, respectively.

$\text{Expt}^{\text{F-IND-CPA},b}_{\mathsf{Ch},N,\mathcal{A}}(1^\lambda)$:
01 $G \leftarrow (\emptyset, \emptyset, \emptyset)$
02 $Q[\,] \leftarrow \emptyset$
03 $(st_1, \ldots, st_N) \leftarrow_\$ \mathsf{Init}(1^\lambda, N)$
04 $b' \leftarrow_\$ \mathcal{A}^{\mathcal{O}_{\mathsf{LoR}}, \mathcal{O}^*_{\mathsf{Recv}}}(1^\lambda)$
05 Terminate with b'

$\mathcal{O}_{\mathsf{LoR}}(i, ad, m_0, m_1)$:
06 Require $|m_0| = |m_1|$
07 $(st_i, c) \leftarrow_\$ \mathsf{Send}(st_i, ad, m_b)$
08 $G \leftarrow G + (\mathtt{S}, i)$ with v
09 $Q[v] \leftarrow (ad, c)$
10 Return c to $\mathcal{A}$

$\mathcal{O}^*_{\mathsf{Recv}}(i, j, ad, c)$:
11 $G \leftarrow G + (\mathtt{R}, i, j)$ with v
12 If $G = \perp \vee (ad, c) \neq Q[\omega(v)]$:
13 Terminate with 0
14 Else:
15 $(st_i, m) \leftarrow_\$ \mathsf{Recv}(st_i, j, ad, c)$
16 Return $\diamond$ to $\mathcal{A}$

$\text{Expt}^{\text{F-IND-CCA},b}_{\mathsf{Ch},N,\mathcal{A}}(1^\lambda)$:
17 $G \leftarrow (\emptyset, \emptyset, \emptyset)$
18 $Q[\,] \leftarrow \emptyset$
19 $\mathsf{active}_1 \leftarrow \cdots \leftarrow \mathsf{active}_N \leftarrow 0$
20 $(st_1, \ldots, st_N) \leftarrow_\$ \mathsf{Init}(1^\lambda, N)$
21 $b' \leftarrow_\$ \mathcal{A}^{\mathcal{O}_{\mathsf{LoR}}, \mathcal{O}^*_{\mathsf{Recv}}}(1^\lambda)$
22 Terminate with b'

$\mathcal{O}_{\mathsf{LoR}}(i, ad, m_0, m_1)$:
23 Require $|m_0| = |m_1|$
24 $(st_i, c) \leftarrow_\$ \mathsf{Send}(st_i, ad, m_b)$
25 If $\mathsf{active}_i = 0$:
26 $G \leftarrow G + (\mathtt{S}, i)$ with v
27 $Q[v] \leftarrow (ad, c)$
28 Return c to $\mathcal{A}$

$\mathcal{O}^*_{\mathsf{Recv}}(i, j, ad, c)$:
29 $G' \leftarrow G + (\mathtt{R}, i, j)$ with v
30 If $G' = \perp \vee (ad, c) \neq Q[\omega(v)]$:
31 $\mathsf{active}_i \leftarrow 1$
32 $(st_i, m) \leftarrow_\$ \mathsf{Recv}(st_i, j, ad, c)$
33 If $\mathsf{active}_i = 1$:
34 Return m to $\mathcal{A}$
35 Else:
36 $G \leftarrow G'$
37 Return $\diamond$ to $\mathcal{A}$

Figure 5.2: *Indistinguishability experiments for FIFO channels. We assume that once a query results in state st_i being set to $\perp$, then no further queries for that participant are accepted; this is without loss of generality, as the channel algorithms would always reject for that participant. We further assume $(i, j) \in [\![1 \,..\, N]\!]$, $ad \in \mathcal{AD}$, $m_0, m_1 \in \mathcal{M}$, and $c \in \mathcal{C}$ for all such values provided by the adversary. When writing '$G + x$ with v' we use v as a placeholder for the node that is newly added to the FIFO graph G in case the operation $G + x$ does not fail. We use sending actions as indices for the associative array $Q[\,]$. A flag active_i per participant is kept to register the participants that are affected by an active measure of the adversary (in which case $\mathsf{active}_i = 1$).*

In the following we give rationale about our experiments.

Chosen-plaintext adversaries. Consider first the F-IND-CPA game (Figure 5.2) which models confidentiality against passive adversaries. For simplicity, ignore the associated data for now. Our formalization loosely follows the *left-or-right* approach for stateful encryption of Bellare *et al.* [BKN02]. In more detail, the oracle $\mathcal{O}_{\mathsf{LoR}}$ takes a participant identifier i and two messages, m_0 and m_1, and returns the ciphertext c obtained by invoking the Send algorithm for the participant indicated by i, on input m_b, where b is a challenge bit the adversary has to guess. Importantly, the participant's state, st_i, is maintained between queries. The oracle $\mathcal{O}^*_{\mathsf{Recv}}$ gives access to the Recv algorithm. Note, however, that it does not respond with the recovered message m, but instead with the special suppression symbol $\diamond$. The reason why a receiving oracle, although having no useful output, is still meaningful in the experiment is that it allows

the adversary to advance the state of participants (this is crucial in the considered setting where participants are both senders and receivers).[3]

Observe that the F-IND-CPA notion shall protect against passive adversaries only, i.e., against attacks in which all ciphertexts presented to $\mathcal{O}^*_{\mathsf{Recv}}$ are faithfully forwarded from the $\mathcal{O}_{\mathsf{LoR}}$ oracle. More precisely, a passive attack in a FIFO network requires that no bit of any ciphertext be flipped, ciphertexts not be replayed, and ciphertexts not be reordered; the latter particularly includes the FIFO order of actions. The task of lines 12–13 is to ensure that the attack remains passive, and to otherwise abort the experiment without giving an advantage to the adversary. To implement this, we record the actions carried out throughout the experiment in a FIFO graph G (lines 01, 08 and 11); in line 09, for each sending action we further record the corresponding ciphertexts in the associative array $Q[\,]$. Note that the condition for passiveness (line 12) requires that the FIFO order of events not be violated (this is tested via $G \neq \bot$) and that the queried ciphertext c be identical to the ciphertext associated to the sending action $\omega(v)$ that corresponds with the current receiving action v. The actual experiment includes an associated data field *ad* that the adversary can choose. Note that in this case passive behavior also requires that the *ad* strings remain consistent (lines 09 and 12).

Chosen-ciphertext adversaries. Let us next discuss the F-IND-CCA experiment (Figure 5.2) which extends F-IND-CPA to also handle active attacks on confidentiality. We introduce a variable active_i for each participant $i \in [1\,..\,N]$ that indicates whether an active measure of the adversary against participant i occurred (line 31, see also the discussion on F-IND-CPA above). If the participant was not exposed to active behavior before, similarly to the F-IND-CPA game, the message output is suppressed; this is indicated by returning $\diamond$ to $\mathcal{A}$ (line 37). Otherwise, the message is given to the adversary (line 34).[4] Observe that our game specification ensures that in G only the passive part of an attack is recorded (indeed, here G records only the 'passive part' of the communication schedule by the adversary and it is used as a mean to determine when the output of Recv shall be suppressed).

In Figure 5.3 we specify the experiments of *integrity of plaintexts* (F-INT-PTXT) and *of ciphertexts* (F-INT-CTXT) for FIFO channels.

Definition 22 (Integrity for FIFO channels)**.** *For* $\text{ATK} \in \{\text{PTXT}, \text{CTXT}\}$ *we say that a FIFO channel* Ch *offers* ATK-integrity *if for all efficient adversaries* $\mathcal{A}$ *and all polynomial* $N = N(\lambda)$ *the following advantage function is negligible,*

$$\mathbf{Adv}^{\text{F-INT-ATK}}_{\mathsf{Ch},N,\mathcal{A}}(\lambda) := \left|\Pr\left[\text{Expt}^{\text{F-INT-ATK}}_{\mathsf{Ch},N,\mathcal{A}}(1^\lambda) = 1\right]\right| \quad .$$

*We abbreviate the notions of integrity of plaintexts (*PTXT*-integrity) and of integrity of ciphertexts (*CTXT*-integrity) for FIFO channels by writing F-INT-PTXT and F-INT-CTXT, respectively.*

We move on to describing our experiments for integrity of ciphertexts and messages, F-INT-PTXT and F-INT-CTXT (Figure 5.3). They are similar to the F-IND-CPA and F-IND-CCA experiments described above, but without the mechanisms for left-or-right sending and

[3] In works like [BKN02] that focus on channels with either-sender-or-receiver functionality, in the definition of confidentiality against passive adversaries a receiving oracle is redundant and thus not annotated in the experiment. We give an example for why in our case it is necessary: Assume a channel in which the first Recv invocation of a participant makes all later Send invocations of the same participant append vital key material to its ciphertext output. Such a scheme is clearly not secure against passive adversaries but, in a model that lacks a receiving oracle, the corresponding attack cannot be expressed.

[4] This is in line with notions of stateless encryption (where requesting the decryption of arbitrary ciphertexts is allowed with the exception of challenge ciphertexts) and stateful encryption [BKN02] (where the decryption oracle becomes functional once the sequence of received ciphertexts gets 'out-of-sync').

$\mathrm{Expt}^{\text{F-INT-PTXT}}_{\mathsf{Ch},N,\mathcal{A}}(1^\lambda)$:
01 $G \leftarrow (\emptyset,\emptyset,\emptyset)$
02 $Q[\,] \leftarrow \emptyset$
03 $(st_1,\ldots,st_N) \leftarrow_\$ \mathsf{Init}(1^\lambda, N)$
04 $\mathcal{A}^{\mathcal{O}_{\mathsf{Send}},\mathcal{O}_{\mathsf{Recv}}}(1^\lambda)$
05 Terminate with 0

If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Send}}(i,ad,m)$:
06 $(st_i,c) \leftarrow_\$ \mathsf{Send}(st_i,ad,m)$
07 $G \leftarrow G + (\mathtt{S},i)$ with v
08 $Q[v] \leftarrow (ad,m)$
09 Return c to $\mathcal{A}$

If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Recv}}(i,j,ad,c)$:
10 $(st_i,m) \leftarrow_\$ \mathsf{Recv}(st_i,j,ad,c)$
11 If $m = \bot$: Return $\bot$ to $\mathcal{A}$
12 Else:
13 $\quad G \leftarrow G + (\mathtt{R},i,j)$ with v
14 $\quad$ If $G = \bot \vee (ad,m) \neq Q[\omega(v)]$:
15 $\quad\quad$ Terminate with 1
16 $\quad$ Return m to $\mathcal{A}$

$\mathrm{Expt}^{\text{F-INT-CTXT}}_{\mathsf{Ch},N,\mathcal{A}}(1^\lambda)$:
17 $G \leftarrow (\emptyset,\emptyset,\emptyset)$
18 $Q[\,] \leftarrow \emptyset$
19 $(st_1,\ldots,st_N) \leftarrow_\$ \mathsf{Init}(1^\lambda, N)$
20 $\mathcal{A}^{\mathcal{O}_{\mathsf{Send}},\mathcal{O}_{\mathsf{Recv}}}(1^\lambda)$
21 Terminate with 0

If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Send}}(i,ad,m)$:
22 $(st_i,c) \leftarrow_\$ \mathsf{Send}(st_i,ad,m)$
23 $G \leftarrow G + (\mathtt{S},i)$ with v
24 $Q[v] \leftarrow (ad,c)$
25 Return c to $\mathcal{A}$

If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Recv}}(i,j,ad,c)$:
26 $(st_i,m) \leftarrow_\$ \mathsf{Recv}(st_i,j,ad,c)$
27 If $m = \bot$: Return $\bot$ to $\mathcal{A}$
28 Else:
29 $\quad G \leftarrow G + (\mathtt{R},i,j)$ with v
30 $\quad$ If $G = \bot \vee (ad,c) \neq Q[\omega(v)]$:
31 $\quad\quad$ Terminate with 1
32 $\quad$ Return m to $\mathcal{A}$

Figure 5.3: *Integrity experiments for FIFO channels. We assume that once a query results in state st_i being set to $\bot$, then no further queries for that participant are accepted. We further assume $(i,j) \in [\![1\,..\,N]\!]$, $ad \in \mathcal{AD}$, $m \in \mathcal{M}$, and $c \in \mathcal{C}$ for all such values provided by the adversary. When writing '$G + x$ with v' we use v as a placeholder for the node that is newly added to the FIFO graph G in case the operation $G + x$ does not fail. We use sending actions as indices for the associative array $Q[\,]$.*

suppressed receiving. The adversary is deemed successful if it exhibits active behavior that is not identified as such by the Recv algorithm. More precisely, an attack on plaintext integrity (F-INT-PTXT) is successful if a pair of ciphertext and associated data processed by Recv produces a valid message and either this message or the associated data is not genuine, or they are delivered in the wrong order (lines 14–15). Our definition of ciphertext integrity (F-INT-CTXT) is similar: here, beside the delivery in the right causal order, this time the combinations of ciphertext and associated data accepted by the Recv algorithm need to be genuine. Note that this is in line with established notions of stateful authenticated encryption [BKN02] where, to capture replay and reordering attacks, ciphertexts need to be delivered to $\mathcal{O}_{\mathsf{Recv}}$ in precisely the same order in which they are generated by the $\mathcal{O}_{\mathsf{Send}}$ oracle. Our integrity notions naturally generalize reordering attacks to the setting of FIFO networks in which the notion of reordering is broader.

Finally, as a technical note, observe that while generally in our integrity games the graph G keeps track of the events occurring in the experiment, it does not record the deliveries which are rejected by Recv. This means that G does not reflect the entire communication scheduled by the adversary. In fact, the graph G here serves as tool to detect whether $\mathcal{A}$ violates the FIFO property. This is in line with the confidentiality experiments: there, an action is recorded in G only if the corresponding query is declared 'passive' (further, to ensure that the attack stays passive, the F-IND-CPA game penalizes the adversary if an active query is ever posed).

5.4 Relations Among Notions

We study the relations among our security notions for FIFO channels. A close inspection of the experiments from Figures 5.2 and 5.3 shows that confidentiality against active adversaries implies confidentiality against passive adversaries, and that ciphertext integrity implies message

integrity. Intuitively, F-IND-CCA $\Longrightarrow$ F-IND-CPA and F-INT-CTXT $\Longrightarrow$ F-INT-PTXT. The converse implications do not hold.[5] This is in line with corresponding results for stateless encryption [BDJR97] and stateful encryption [BKN02].

The F-INT-CTXT notions ensures that the accepting invocations of Send and Recv do not involve manipulated ciphertexts or associated data and follow the corresponding order relation. As a cryptographic tool they aim at restricting the adversary to passive behavior. Thus, intuitively, if a channel offers ciphertext integrity then the active and passive notions of confidentiality imply each other. Put differently, if a FIFO channel offers F-INT-CTXT and F-IND-CPA security then it is also F-IND-CCA-secure. Theorem 5 captures this important result. It is akin to findings in the contexts of stateless and stateful authenticated encryption [BN00, BKN02].

Theorem 5 (F-IND-CPA $\wedge$ F-INT-CTXT $\Longrightarrow$ F-IND-CCA)**.** *Let* Ch *be a FIFO channel that offers integrity of ciphertexts and indistinguishability under chosen-message attacks. Then* Ch *also offers indistinguishability under chosen-ciphertext attacks. More precisely, for every efficient adversary* $\mathcal{A}$ *there exist efficient adversaries* $\mathcal{B}$ *and* $\mathcal{C}$ *such that*

$$\mathbf{Adv}^{\text{F-IND-CCA}}_{\mathsf{Ch},N,\mathcal{A}}(\lambda) \leq 2 \cdot \mathbf{Adv}^{\text{F-INT-CTXT}}_{\mathsf{Ch},N,\mathcal{B}}(\lambda) + \mathbf{Adv}^{\text{F-IND-CPA}}_{\mathsf{Ch},N,\mathcal{C}}(\lambda) \ .$$

$\mathcal{B}_b^{\mathcal{O}_{\mathsf{Send}},\mathcal{O}_{\mathsf{Recv}}}(1^\lambda)$:
01 $b' \leftarrow_{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{LoR}},\mathcal{O}^*_{\mathsf{Recv}}}(1^\lambda)$
02 Terminate with 0

$\mathcal{O}_{\mathsf{LoR}}(i, ad, m_0, m_1)$:
03 Require $|m_0| = |m_1|$
04 $c \leftarrow_{\$} \mathcal{O}_{\mathsf{Send}}(i, ad, m_b)$
05 Return c to $\mathcal{A}$

$\mathcal{O}^*_{\mathsf{Recv}}(i, j, ad, c)$:
06 $m \leftarrow_{\$} \mathcal{O}_{\mathsf{Recv}}(i, j, ad, c)$
07 If $m = \bot$:
08 return $\bot$ to $\mathcal{A}$
09 Else:
10 Return $\diamond$ to $\mathcal{A}$

$\mathcal{C}^{\mathcal{O}_{\mathsf{LoR}},\mathcal{O}^*_{\mathsf{Recv}}}(1^\lambda)$:
11 $G \leftarrow (\emptyset, \emptyset, \emptyset), Q[\,] \leftarrow \emptyset$
12 $\mathsf{active}_1 \leftarrow \cdots \leftarrow \mathsf{active}_N \leftarrow 0$
13 $b' \leftarrow_{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{LoR}},\mathcal{O}^*_{\mathsf{Recv}}}(1^\lambda)$
14 Terminate with b'

$\mathcal{O}_{\mathsf{LoR}}(i, ad, m_0, m_1)$:
15 Require $|m_0| = |m_1|$
16 $c \leftarrow_{\$} \mathcal{O}_{\mathsf{LoR}}(i, ad, m_0, m_1)$
17 If $\mathsf{active}_i = 0$:
18 $G \leftarrow G + (\mathtt{S}, i)$ with v
19 $Q[v] \leftarrow (ad, c)$
20 Return c to $\mathcal{A}$

$\mathcal{O}^*_{\mathsf{Recv}}(i, j, ad, c)$:
21 $G' \leftarrow G + (\mathtt{R}, i, j)$ with v
22 If $G' = \bot \vee (ad, c) \neq Q[\omega(v)]$:
23 $\mathsf{active}_i \leftarrow 1$
24 If $\mathsf{active}_i = 1$:
25 Return $\bot$ to $\mathcal{A}$
26 Else:
27 $\diamond \leftarrow \mathcal{O}^*_{\mathsf{Recv}}(i, j, ad, c)$
28 $G \leftarrow G'$
29 Return $\diamond$ to $\mathcal{A}$

Figure 5.4: *Security reductions* $\mathcal{B}_b$ *and* $\mathcal{C}$ *used in the proof for F-IND-CPA* $\wedge$ *F-INT-CTXT* $\Longrightarrow$ *F-IND-CCA (see Theorem 5).*

Proof. Let $\mathrm{E}^{0,b}_{\mathcal{A}}$ denote the F-IND-CCA experiment from Figure 5.2 against Ch, and let $\Pr[\mathrm{E}^{0,b}_{\mathcal{A}}]$ be a shortcut for the probability $\Pr[\mathrm{E}^{0,b}_{\mathcal{A}}(1^\lambda) = 1]$. We proceed via game-hopping. Let us define experiment $\mathrm{E}^{1,b}_{\mathcal{A}}$ from $\mathrm{E}^{0,b}_{\mathcal{A}}$ by replacing line 34 (on page 57) with the instruction 'Terminate with 0'. For $b \in \{0, 1\}$ let bad^b denote the events that, during an execution of either game $\mathrm{E}^{0,b}_{\mathcal{A}}$ or game $\mathrm{E}^{1,b}_{\mathcal{A}}$, the adversary triggers the instruction of line 34 (on page 57). By construction we have $\Pr[\mathrm{E}^{0,b}_{\mathcal{A}} \wedge \neg bad^b] = \Pr[\mathrm{E}^{1,b}_{\mathcal{A}} \wedge \neg bad^b]$, and hence we can bound $|\Pr[\mathrm{E}^{0,b}_{\mathcal{A}}] - \Pr[\mathrm{E}^{1,b}_{\mathcal{A}}]| \leq \Pr[bad^b]$.

[5] This is best seen by considering channels that are constructed from other channels by appending a redundant zero-bit to each ciphertext; this transformation preserves the F-IND-CPA and the F-INT-CTXT notions but allows easily breaking the corresponding F-IND-CCA and F-INT-CTXT notions.

Now we build two adversaries, $\mathcal{B}_0$ and $\mathcal{B}_1$, whose CTXT-advantage is related to the probability that $\mathcal{A}$ triggers events bad^0 and bad^1 respectively. Adversary $\mathcal{B}_b$ emulates a left-or-right oracle using its own sending oracle: if $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{LoR}}$ on input (i, ad, m_0, m_1) then $\mathcal{B}_b$ asks (i, ad, m_b) to $\mathcal{O}_{\mathsf{Send}}$ and forwards the oracle answer to $\mathcal{A}$; similarly, $\mathcal{B}_b$ uses oracle $\mathcal{O}_{\mathsf{Recv}}$ to answer queries that $\mathcal{A}$ poses to $\mathcal{O}^*_{\mathsf{Recv}}$. A full specification of $\mathcal{B}_b$'s code is given in Figure 5.4. Observe that $\mathcal{B}_b$ performs a perfect simulation of game $\mathrm{E}^{1,b}_{\mathcal{A}}$ as long as event bad^b does not occur; however, if bad^b happens then $\mathcal{B}_b$ breaks ciphertext integrity. This implies $\Pr[bad^b] \leq \mathbf{Adv}^{\text{F-INT-CTXT}}_{\mathsf{Ch},N,\mathcal{B}_b}(\lambda)$. Consider now an adversary $\mathcal{B}$ which tosses a coin and then runs $\mathcal{B}_0$ or $\mathcal{B}_1$ according to the outcome. By construction, $\mathcal{B}$'s advantage is the average of $\mathcal{B}_0$ and $\mathcal{B}_1$'s advantages, hence $\mathbf{Adv}^{\text{F-INT-CTXT}}_{\mathsf{Ch},\mathcal{B}}(\lambda) \geq \Pr[bad^0 \wedge d = 0] + \Pr[bad^1 \wedge d = 1] = \frac{1}{2} \cdot \Pr[bad^0] + \frac{1}{2} \cdot \Pr[bad^1]$. We can now derive the following bound for $\mathcal{A}$'s advantage in the original game:

$$\begin{aligned}
\mathbf{Adv}^{\text{F-IND-CCA}}_{\mathsf{Ch},N,\mathcal{A}}(\lambda) &= \left|\Pr\left[\mathrm{E}^{0,1}_{\mathcal{A}}\right] - \Pr\left[\mathrm{E}^{0,0}_{\mathcal{A}}\right]\right| \\
&\leq \left|\Pr\left[\mathrm{E}^{0,1}_{\mathcal{A}}\right] - \Pr\left[\mathrm{E}^{1,1}_{\mathcal{A}}\right]\right| + \left|\Pr\left[\mathrm{E}^{1,1}_{\mathcal{A}}\right] - \Pr\left[\mathrm{E}^{1,0}_{\mathcal{A}}\right]\right| + \left|\Pr\left[\mathrm{E}^{1,0}_{\mathcal{A}}\right] - \Pr\left[\mathrm{E}^{0,0}_{\mathcal{A}}\right]\right| \\
&\leq \Pr\left[bad^1\right] + \left|\Pr\left[\mathrm{E}^{1,1}_{\mathcal{A}}\right] - \Pr\left[\mathrm{E}^{1,0}_{\mathcal{A}}\right]\right| + \Pr\left[bad^0\right] \\
&\leq 2 \cdot \mathbf{Adv}^{\text{F-INT-CTXT}}_{\mathsf{Ch},N,\mathcal{B}}(\lambda) + \left|\Pr\left[\mathrm{E}^{1,1}_{\mathcal{A}}\right] - \Pr\left[\mathrm{E}^{1,0}_{\mathcal{A}}\right]\right| \ .
\end{aligned}$$

It remains to prove that chosen-message power suffices to perform a faithful simulation of game $\mathrm{E}^{1,b}_{\mathcal{A}}$. To this end we construct an adversary $\mathcal{C}$ (Figure 5.4) that emulates game $\mathrm{E}^{1,b}_{\mathcal{A}}$ using the oracles provided by the F-IND-CPA game. Briefly, $\mathcal{C}$ maintains a communication graph G to keep track of the adversarially scheduled communication, relays $\mathcal{A}$'s sending queries to its left-or-right oracle ($\mathcal{O}_{\mathsf{LoR}}$ executes the same instructions in the F-IND-CPA and F-IND-CCA games) and registers pairs (ad, c) corresponding to each sending query. If $\mathcal{A}$ asks a receiving query then $\mathcal{C}$ determines if this query is passive (a.k.a. in-sync) or active (a.k.a. out-of-sync): in the first case $\mathcal{C}$ forwards the query to its receiving oracle (to advance the participant's state) and returns the suppression symbol '$\diamond$' to $\mathcal{A}$; in the second case, it answers directly by returning the error symbol '$\perp$'. To detect active queries, $\mathcal{C}$ keeps for each user i a flag $\mathsf{active}_i \leftarrow 0$ (as done in the F-IND-CCA game) and sets it to $\mathsf{active}_i \leftarrow 1$ when the first receiving query (i, j, ad, c) is made such that either $G' + (\mathtt{R}, i, j) = \perp$ or the current pair (ad, c) does not match the correspondingly sent pair. We claim that $\mathcal{C}$ performs a perfect simulation of game $\mathrm{E}^{1,b}_{\mathcal{A}}$. Indeed, an adversary $\mathcal{A}$ that is not penalized in game $\mathrm{E}^{1,b}_{\mathcal{A}}$ will never cause premature termination of the game. The reduction $\mathcal{C}$ is then left with two possibilities for answering any receiving query posed by $\mathcal{A}$: either the query is 'active' and should be rejected, or it is 'passive' and the algorithm's answer shall be suppressed. $\mathcal{C}$ precisely detects which one is the case and reacts accordingly. We finally obtain the desired inequality:

$$\mathbf{Adv}^{\text{F-IND-CCA}}_{\mathsf{Ch},N,\mathcal{A}}(\lambda) \leq 2 \cdot \mathbf{Adv}^{\text{F-INT-CTXT}}_{\mathsf{Ch},N,\mathcal{B}}(\lambda) + \mathbf{Adv}^{\text{F-IND-CPA}}_{\mathsf{Ch},N,\mathcal{C}}(\lambda) \ .$$

□

5.5 Unidirectional Channels

Later in this chapter we will see how to build FIFO channels from unidirectional channels. For the sake of completeness we specify here syntax, correctness and security for unidirectional channels. Let us first formalize the intuitive concept of *unidirectional communication* (between two participants) using the graph terminology developed in Chapter 4. Throughout this section we assume without further notice that all communication graphs involve two participants, Alice and Bob. Intuitively speaking, a left-to-right communication graph represents a conversation

where Alice speaks (and doesn't listen) while Bob only listens (and never speaks); in a right-to-left communication graph the roles are exchanged.

Definition 23 (Unidirected communication graphs)**.** *Let $G = (V, \leq, \chi)$ be a communication graph, let V_1 and V_2 be the sets of actions performed by Party 1, respectively, by Party 2, and let V^S and V^R be the sets of sending actions, respectively, receiving actions in V. We say that G is* left-to-right *if $V^S = V_1$ and $V^R = V_2$. Analogously, G is* right-to-left *if $V_1 = V^R$ and $V_2 = V^S$. We say that G is* unidirected *if it is either left-to-right or right-to-left.*

A unidirectional channel is a restricted two-party FIFO channel that lets one participant, Alice, only invoke algorithm Send and the other participant, Bob, only invoke Recv.

In the rest of this section we provide functional specifications and security requirements for unidirectional channels. We stress that a unidirectional channel is a special case of a FIFO channel; thus, the corresponding notions are neither novel nor surprising. However, the reduced complexity of the unidirected setting leads to a simplified notation which may help the reader to bridge the stateful authenticated encryption notions, which are restricted to one direction of communication, and our notions for FIFO channels. Moreover, defining unidirectional channels as a special case of FIFO channels provides some validation for our notions of broadcast channels.

Definition 24 (Syntax and correctness of unidirectional channels)**.** *A* unidirectional channel $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *with associated data space $\mathcal{AD}$, message space $\mathcal{M}$, ciphertext space $\mathcal{C}$, and state space $\mathcal{S}$, consists of efficient probabilistic algorithms as follows:*

- Init*. This algorithm takes as input a security parameter 1^λ and outputs initial sending state and receiving state $st_S, st_R \in \mathcal{S}$. We write $(st_S, st_R) \leftarrow_\$ \mathsf{Init}(1^\lambda)$.*
- Send*. This algorithm takes as input a state $st_S \in \mathcal{S}$, associated data $ad \in \mathcal{AD}$, and a message $m \in \mathcal{M}$, and outputs a state $st'_S \in \mathcal{S}$ and a ciphertext $c \in \mathcal{C}$. We write $(st'_S, c) \leftarrow_\$ \mathsf{Send}(st_S, ad, m)$.*
- Recv*. This algorithm takes as input a state $st_R \in \mathcal{S}$, associated data $ad \in \mathcal{AD}$, and a ciphertext $c \in \mathcal{C}$, and outputs a state $st'_R \in \mathcal{S}$ and a message $m \in \mathcal{M}$ or $m = \perp$. We write $(st'_R, m) \leftarrow_\$ \mathsf{Recv}(st_R, ad, c)$.*

Let $G = (V, \leq, \chi)$ be a left-to-right graph and for $n = |V|$ assume an enumeration $e\colon [1\,..\,n] \to V$ of $(V, \leq)$. Let $\alpha\colon V^S \to \mathcal{AD}$ and $\mu\colon V^S \to \mathcal{M}$ denote arbitrary assignments. Denote by $d[\,]$ and $m[\,]$ associative arrays that map $V^S \to \mathcal{C}$ and $V^R \to \mathcal{M}$, respectively. Consider the following procedure:

---

01 Initialize states $(st_{S,0}, st_{R,0}) \leftarrow_\$ \mathsf{Init}(1^\lambda)$
02 Process the actions $v \in V$ in order $v_1 = e(1), \ldots, v_n = e(n)$ according to the rules:
03 – if $\chi(v_i) = (\mathtt{S}, *)$ then $(st_{S,i}, c[v]) \leftarrow_\$ \mathsf{Send}(st_{S,i-1}, \alpha(v), \mu(v))$
04 – if $\chi(v_i) = (\mathtt{R}, *, *)$ then $(st_{R,i}, m[v]) \leftarrow_\$ \mathsf{Recv}(st_{R,i-1}, \alpha(\omega(v)), c[\omega(v)])$

We say that channel Ch is correct *if for all left-to-right FIFO graph $G \in \mathcal{G}_{\text{FIFO}}$, all choices of e, α, μ, and for all randomnesses of the Init, Send, and Recv algorithms, the Recv algorithm correctly recovers all sent messages, i.e., $m[v] = \mu(\omega(v))\, \forall v \in V^R$.*

Remark 4 (Alternative correctness condition *à la* BKN [BKN02])*.* Observe that the correctness condition for unidirectional channels is equivalent to the corresponding condition for stateful encryption [BKN02], as long as one ignores the associated data. To see why, note that for every

unidirected FIFO graph $G = (V, \leq, \chi)$ the visit of V that first walks along the actions in V^S and then the actions in V^R, according to the local order $\prec_\ell$, is a valid enumeration of $(V, \leq)$. Given this, we can rephrase the correctness condition for unidirected channels as follows: for every choice of the randomness for the Init, Send, and Recv algorithms, every initial state pair $(st_{S,0}, st_{R,0}) \leftarrow_\$ \mathsf{Init}(1^\lambda)$, every associated data tuple $(ad_1, \ldots, ad_s) \in \mathcal{AD}^*$, every message tuple $(m_1, \ldots, m_s) \in \mathcal{M}^*$, and every ciphertext tuple $(c_1, \ldots, c_s) \in \mathcal{C}^*$ generated by invoking sequentially $(st_{S,1}, c_1) \leftarrow_\$ \mathsf{Send}(st_{S,0}, ad_1, m_1), \ldots, (st_{S,s}, c_s) \leftarrow_\$ \mathsf{Send}(st_{S,s-1}, ad_s, m_s)$, we have that every received message sequence $(m'_1, \ldots, m'_r) \in \mathcal{M}^*$ for $r \leq s$ generated by invoking sequentially $(st_{R,1}, m'_1) \leftarrow_\$ \mathsf{Recv}(st_{R,0}, ad_1, c_1), \ldots, (st_{R,r}, m'_r) \leftarrow_\$ \mathsf{Recv}(st_{R,r-1}, ad_r, c_r)$, is a prefix of the originally sent message tuple, i.e., $(m'_1, \ldots, m'_r) = (m_1, \ldots, m_r)$.

$\mathrm{Expt}^{\text{IND-CCA}}_{\mathsf{Ch},\mathcal{A}}(1^\lambda)$:
01 $s \leftarrow r \leftarrow 0$
02 $\mathsf{active} \leftarrow 0$
03 $(st_S, st_R) \leftarrow_\$ \mathsf{Init}(1^\lambda)$
04 $b' \leftarrow_\$ \mathcal{A}^{\mathcal{O}_{\mathsf{LoR}}, \mathcal{O}^*_{\mathsf{Recv}}}(1^\lambda)$
05 Terminate with b'

$\mathcal{O}_{\mathsf{LoR}}(ad, m_0, m_1)$:
06 Require $|m_0| = |m_1|$
07 $s \leftarrow s + 1$
08 $(st_S, c) \leftarrow_\$ \mathsf{Send}(st_S, ad, m_b)$
09 $ad_s \leftarrow ad, c_s \leftarrow c$
10 Return c to $\mathcal{A}$

$\mathcal{O}^*_{\mathsf{Recv}}(ad, c)$:
11 $r \leftarrow r + 1$
12 If $r > s \vee (ad, c) \neq (ad_r, c_r)$:
13 $\quad \mathsf{active} \leftarrow 1$
14 $(st_R, m) \leftarrow_\$ \mathsf{Recv}(st_R, ad, c)$
15 If $\mathsf{active} = 1$:
16 $\quad$ Return m to $\mathcal{A}$
17 Else:
18 $\quad$ Return $\diamond$ to $\mathcal{A}$

$\mathrm{Expt}^{\text{INT-CTXT}}_{\mathsf{Ch},\mathcal{A}}(1^\lambda)$:
19 $s \leftarrow r \leftarrow 0$
20 $(st_S, st_R) \leftarrow_\$ \mathsf{Init}(1^\lambda)$
21 $\mathcal{A}^{\mathcal{O}_{\mathsf{Send}}, \mathcal{O}_{\mathsf{Recv}}}(1^\lambda)$
22 Terminate with 0

$\mathcal{O}_{\mathsf{Send}}(ad, m)$:
23 $(st_S, c) \leftarrow_\$ \mathsf{Send}(st_S, ad, m)$
24 $s \leftarrow s + 1$
25 $ad_s \leftarrow ad, \boxed{c_s \leftarrow c}$
26 Return c to $\mathcal{A}$

$\mathcal{O}_{\mathsf{Recv}}(ad, c)$:
27 $(st_R, m) \leftarrow \mathsf{Recv}(st_R, ad, c)$
28 $r \leftarrow r + 1$
29 If $m = \perp$:
30 $\quad$ Return $\perp$ to $\mathcal{A}$
31 Else:
32 $\quad$ If $r > s \vee \boxed{(ad, c) \neq (ad_r, c_r)}$:
33 $\quad\quad$ Terminate with 1
34 $\quad$ Return m to $\mathcal{A}$

Figure 5.5: *Confidentiality and integrity experiments for unidirectional channels. We can derive the IND-CPA game from the IND-CCA game by ignoring the receiving oracle. Similarly, we can obtain the INT-PTXT game from the INT-CTXT game by replacing the boxed text in lines 25 and 32 with '$m_s \leftarrow m$' and '$(ad, m) \neq (ad, m_r)$'.*

We specify confidentiality and integrity experiments for unidirectional channels in Figure 5.5. Note that we deviate from the FIFO experiments (Figures 5.2 and 5.3) by simplifying the mechanism to detect if the attacker exhibits active behavior. More precisely, instead of keeping an auxiliary communication graph G and checking for all receiving query if the corresponding addition is admissible in a FIFO sense, here we simply compare two counters s and r (for sent and received messages). As a consequence of Lemma 1(a) and Definition 16 (which jointly imply that adding a receiving action $(\mathtt{R}, i, j)$ to a FIFO graph is possible if and only Party j has sent at least as often as Party i has received from Party j) the two approaches are equivalent. Indeed, recall that for a unidirected FIFO graph a receiving action is admissible if and only if sufficiently many sending actions were already performed ($r \leq s$ in the notation of the experiments from Figure 5.5). We opted for this change to make the experiments for unidirectional FIFO channels closer to the ones from [BKN02] and ease the comparison between their model and ours.

As a validation for our FIFO security notions, observe indeed that the security games from Figure 5.5 are equivalent to the IND-CPA, IND-CCA, INT-PTXT and INT-CTXT experiments for stateful authenticated encryption (Figure 2.2 on page 14, or [BKN02]). Except for the *ad* field, any difference is purely syntactical. With other words, our security notions for FIFO channels if restricted to the unidirectional case collapse to stateful AE(AD).

5.6 Constructions

FIFO channels are easy to construct from standard cryptographic primitives. In this section we propose two design strategies to realize FIFO channels from simpler, symmetric building blocks. The first is based on AEAD. The second relies on unidirectional channels ('essentially' stateful AEAD) and realizes a bidirectional channel, i.e., it only supports two participants. In fact, the latter formalizes the bidirectional channel design underlying (the cryptographic core of) the TLS Record Protocol and the SSH Binary Packet Protocol. Finally, we establish under which conditions the proposed constructions provably achieve the strongest security guarantees for FIFO channels, namely F-IND-CCA and F-INT-CTXT security.

5.6.1 FIFO Channels from AEAD

We describe how to build a FIFO channel from any AEAD scheme. The setup is that all participants share the same symmetric AEAD key K and manage N counters each, $\mathrm{ctr}_1, \ldots, \mathrm{ctr}_N$. Participant $i \in [1\,..\,N]$ registers in ctr_i the number of messages it has sent so far and, in the remaining counters ctr_j for $j \neq i$ it stores how many messages it has received from user j so far. Now, when participant $i \in [1\,..\,N]$ wants to send a message m with associated data field ad, it increments its counter ctr_i and invokes the encryption routine on input key K, message m, and associated data string $i \,\|\, \mathrm{ctr}_i \,\|\, ad$, obtains an AEAD ciphertext c, and then broadcasts the ciphertext c. For an incoming ciphertext c with alleged originator $j \neq i$, participant i increments ctr_j and AEAD-decrypts ciphertext c with associated data string $j \,\|\, \mathrm{ctr}_j \,\|\, ad$, obtaining a message m which it then outputs, or $\perp$ indicating that an error occurred.

A concise specification of the scheme is given in Figure 5.6.

Construction 2 (FIFO Channels from AEAD)**.** *Let $\mathcal{AD}$ be an associated data space and let $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be an AEAD scheme with associated data space $[1\,..\,N] \times \mathbb{N} \times \mathcal{AD}$. Define the channel $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ as described in Figure 5.6.*

It is immediate to see that Construction 2 fulfills the correctness requirements of a FIFO channel (as in Definition 20). Moreover, as we prove next, AEAD security (see Section 2.3.4 on page 12) of the underlying scheme tightly implies both F-INT-CTXT and F-IND-CPA, thus, by Theorem 5, also F-IND-CCA.

Theorem 6 (Confidentiality of Construction 2)**.** *Let Π be an AEAD scheme and let Ch be the FIFO channel obtained from Π by applying the transformation described in Construction 2. If Π offers indistinguishability under chosen-plaintext attacks, so does Ch is a FIFO sense. More precisely, for every efficient adversary $\mathcal{A}$ attacking the F-IND-CPA property of Ch there exists an efficient adversary $\mathcal{B}$ against the IND-CPA property of Π such that*

$$\mathbf{Adv}^{\text{F-IND-CPA}}_{\mathsf{Ch},N,\mathcal{A}}(\lambda) \leq \mathbf{Adv}^{\text{IND-CPA}}_{\Pi,\mathcal{B}}(\lambda) \ .$$

Proof. Recall that in the F-IND-CPA game the adversary has access to both a left-or-right oracle $\mathcal{O}_{\mathsf{LoR}}$ and a receiving oracle $\mathcal{O}^*_{\mathsf{Recv}}$, where the latter is needed to let the adversary advance the state of participants through receiving actions (without learning the underlying message).

$\mathsf{Init}(1^\lambda, N)$:
01 $K \leftarrow_\$ \mathsf{KeyGen}(1^\lambda)$
02 For $i \leftarrow 1$ to N:
03 $\quad \mathrm{ctr}_1 \leftarrow \cdots \leftarrow \mathrm{ctr}_N \leftarrow 0$
04 $\quad st_i \leftarrow (K, \mathrm{ctr}_1, \ldots, \mathrm{ctr}_N)$
05 Return $(st_1, \ldots, st_N)$

$\mathsf{Send}(st_i, ad, m)$:
06 If $st_i = \bot$:
07 $\quad$ Return $\bot$
08 Parse st_i as $(K, \mathrm{ctr}_1, \ldots, \mathrm{ctr}_N)$
09 $\mathrm{ctr}_i \leftarrow \mathrm{ctr}_i + 1$
10 $ad' \leftarrow i \,\|\, \mathrm{ctr}_i \,\|\, ad$
11 $c \leftarrow_\$ \mathsf{Enc}_K(ad', m)$
12 Return (st_i, c)

$\mathsf{Recv}(st_i, j, ad, c)$:
13 If $st_i = \bot$:
14 $\quad$ Return $\bot$
15 Parse st_i as $(K, \mathrm{ctr}_1, \ldots, \mathrm{ctr}_N)$
16 $\mathrm{ctr}_j \leftarrow \mathrm{ctr}_j + 1$
17 $ad' \leftarrow j \,\|\, \mathrm{ctr}_j \,\|\, ad$
18 $m \leftarrow \mathsf{Dec}_K(ad', c)$
19 If $m = \bot$:
20 $\quad st_i \leftarrow \bot$
21 Return (st_i, m)

Figure 5.6: *Generic construction of a FIFO channel* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *with associated data space* $\mathcal{AD}$ *from any AEAD scheme* $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *with associated data space* $[1 \,..\, N] \times \mathbb{N} \times \mathcal{AD}$.

The reduction $\mathcal{B}$, thus, while being granted by the IND-CPA experiment only chosen-plaintext capabilities, has to process receiving queries and, in particular, update the states of the participants accordingly. This is not a problem. First of all, notice that $\mathcal{B}$ only has to deal with in-sync receiving queries as $\mathcal{A}$ is expected in the F-IND-CPA game to maintain passive behavior. Moreover, the state update performed by Recv when processing an incoming ciphertext c with alleged originator j simply consists in incrementing the counter ctr_j, i.e., it does not require knowledge of any secret. Hence, to perform a sound simulation of the F-IND-CPA game all what $\mathcal{B}$ has to do is: to check that $\mathcal{A}$ poses only in-sync queries; to advance the states of the participants; and to answer left-or-right queries. Intuitively, the reduction can do so by counting for each participant the number of left-or-right and receiving queries that $\mathcal{A}$ requests, by incrementing participants' counters accordingly, and by relaying all receiving queries to the left-or-right oracle provided by the IND-CPA game.

In greater detail, let $\mathcal{B}$ keep counters ctr_{ij} and lists $Q_i[\,]$ for $i, j \in [1 \,..\, N]$, initialized as $\mathrm{ctr}_{ij} \leftarrow 0$ and $Q_i[\,] \leftarrow \emptyset$ for all values of i and j. For every left-or-right query (i, ad, m_0, m_1) that $\mathcal{A}$ poses, $\mathcal{B}$ increments counter ctr_{ii}, invokes the left-or-right oracle $\mathcal{O}_{\mathsf{LoR}}$ on associated data $ad' = i \,\|\, \mathrm{ctr}_{ii} \,\|\, ad$ and message pair (m_0, m_1), gets back an AEAD ciphertext c, registers (ad, c) into $Q_i[\mathrm{ctr}_{ii}]$, and finally returns ciphertext c to $\mathcal{A}$. For every receiving query of the form (i, j, ad, c), the reduction increments counter ctr_{ij} and, to ensures that $\mathcal{A}$ sticks to passive behavior, verifies that $\mathrm{ctr}_{ij} \leq \mathrm{ctr}_{jj}$, i.e., that participant i receives from j at most as often as j performed sending actions[6] and that $(ad, c) = Q_j[\mathrm{ctr}_{ij}]$, i.e., that the received pair of associated data and ciphertext matches the correspondingly sent pair. If not, $\mathcal{B}$ stops the simulation and halts; otherwise, it returns the suppression symbol '$\diamond$' to $\mathcal{A}$. Eventually $\mathcal{A}$ returns a bit b' as a prediction of the hidden bit b: $\mathcal{B}$ outputs b' and halts. We conclude that $\mathcal{A}$ and $\mathcal{B}$ have the same distinguishing advantage. □

[6]By Lemma 1 (a)–(b) and by definition of the FIFO addition '+' (Chapter 4) we know that the only way for $\mathcal{A}$ to violate the FIFO property is to cause for two participants $(i, j) \in [\![1 \,..\, N]\!]$ that i receives from j more often than j sent.

Theorem 7 (Integrity of Construction 2)**.** *Let* Π *be an AEAD scheme and let* Ch *be the FIFO channel obtained from* Π *by applying the transformation described in Construction 2. If* Π *offers integrity of ciphertexts, so does* Ch *in a FIFO sense. More precisely, for every efficient adversary* $\mathcal{A}$ *attacking the F-INT-CTXT property of* Ch *there exists an efficient adversary* $\mathcal{C}$ *against the INT-CTXT property of* Π *such that*

$$\mathbf{Adv}^{\text{F-INT-CTXT}}_{\mathsf{Ch},N,\mathcal{A}}(\lambda) \leq \mathbf{Adv}^{\text{INT-CTXT}}_{\Pi,\mathcal{C}}(\lambda) \ .$$

Proof. The reduction $\mathcal{C}$ can easily emulate sending and receiving oracles $\mathcal{O}_{\mathsf{Send}}$ and $\mathcal{O}_{\mathsf{Recv}}$ for $\mathcal{A}$ using the encryption and decryption oracles $\mathcal{O}_{\mathsf{Enc}}$ and $\mathcal{O}_{\mathsf{Dec}}$ provided by the INT-CTXT game. As the reduction $\mathcal{B}$ described in the proof of Theorem 6 (on page 64), the reduction $\mathcal{C}$ keeps counters ctr_{ij} for $i, j \in [1\,..\,N]$. When $\mathcal{A}$ poses a sending query (i, ad, m), $\mathcal{C}$ queries $\mathcal{O}_{\mathsf{Enc}}$ on associated data $i \,\|\, \mathrm{ctr}_{ii} \,\|\, ad$ and message m, hence returns the oracle answer to $\mathcal{A}$. Similarly, when $\mathcal{A}$ poses a receiving query (i, j, ad, c), $\mathcal{C}$ asks $\mathcal{O}_{\mathsf{Dec}}$ to decrypt c with associated data $i \,\|\, \mathrm{ctr}_{ij} \,\|\, ad$ and then gives $\mathcal{A}$ the ciphertext returned by the oracle. It is immediate to see that $\mathcal{C}$, using its oracles, executes the same instructions of the F-INT-CTXT game. To see that any valid forgery produced by $\mathcal{A}$ in the simulated F-INT-CTXT game corresponds to a valid forgery in the INT-CTXT game, first observe that algorithm Recv accepts only if the AEAD decryption performed internally does (line 18 in Figure 5.6). It remains to prove that a forgery (i, j, ad^*, c^*) for the F-INT-CTXT game corresponds to a forgery (ad'^*, c^*) in the INT-CTXT game. The latter follows from the fact that $\mathcal{C}$ never submits to $\mathcal{O}_{\mathsf{Enc}}$ the same associated data twice: indeed, for every sending query (i, ad, m) posed by $\mathcal{A}$, the reduction asks $\mathcal{O}_{\mathsf{Enc}}$ to encrypt m using associated data $ad' = i \,\|\, \mathrm{ctr}_{ii} \,\|\, ad$, where ctr_{ii} is a counter that increases with every encryption query and, thus, makes the combination $i \,\|\, \mathrm{ctr}_{ii}$ unique. □

5.6.2 FIFO Channels from Unidirectional Channels

The design principle of many widely deployed channel protocols (including TLS and SSH) builds a bidirectional channel using two *independent* unidirectional channels in opposite directions, letting one user send messages through the one channel and receive from the other channel, and vice versa for the second user. We call this paradigm the *canonic composition* of unidirectional channels, and specify its details in Construction 5.7. The bidirectional channel established by the TLS record protocol [DR08] represents a concrete instantiation of the canonic composition where the unidirectional channels use independent keys to secure each direction of communication.

Construction 3 (Canonic Composition)**.** *d Consider two unidirectional channels* $\mathsf{Ch}_1 = (\mathsf{Init}_1, \mathsf{Send}_1, \mathsf{Recv}_1)$ *and* $\mathsf{Ch}_2 = (\mathsf{Init}_2, \mathsf{Send}_2, \mathsf{Recv}_2)$ *with associated data space* $\mathcal{AD}$*, message space* $\mathcal{M}$*, ciphertext space* $\mathcal{C}$*, and state spaces* $\mathcal{S}_1$ *and* $\mathcal{S}_2$ *respectively. Set* $\mathcal{S} = (\mathcal{S}_1 \times \mathcal{S}_2) \cup (\mathcal{S}_2 \times \mathcal{S}_1)$ *and let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be the bidirectional channel with state space* $\mathcal{S}$ *depicted in Figure 5.7.*

It is easy to see that if Ch_1 and Ch_2 are correct in a unidirectional sense then Ch is a correct FIFO channel. Indeed, the FIFO ordering property particularly implies that ciphertexts are delivered according to the sending order, for each sender; thus, the correctness condition is fulfilled for the two unidirectional channels Ch_1 and Ch_2 independently.

We now turn to analyzing the security of the canonic composition. As we show next, Construction 3 generically inherits some of the security properties of its building blocks, but not all of them.

Confidentiality against passive adversaries and integrity are preserved. We claim that if both unidirectional channels Ch_1 and Ch_2 are confidential against *passive* attacks, i.e., chosen-plaintext attacks (IND-CPA), then their canonic composition Ch is confidential against

$\mathsf{Init}(1^\lambda)$:
01 $(st_{S,1}, st_{R,1}) \leftarrow_\$ \mathsf{Init}_1(1^\lambda)$
02 $(st_{S,2}, st_{R,2}) \leftarrow_\$ \mathsf{Init}_2(1^\lambda)$
03 $st_1 \leftarrow (st_{S,1}, st_{R,2})$
04 $st_2 \leftarrow (st_{S,2}, st_{R,1})$
05 Return (st_1, st_2)

$\mathsf{Send}(st, ad, m)$:
06 If $st = \bot$: Return $\bot$
07 Parse st as $(st_{S,i}, st_R)$
08 $(st'_{S,i}, c) \leftarrow_\$ \mathsf{Send}_i(st_{S,i}, ad, m)$
09 $st \leftarrow (st'_{S,i}, st_R)$
10 Return (st, c)

$\mathsf{Recv}(st, ad, c)$:
11 If $st = \bot$: Return $\bot$
12 Parse st as $(st_S, st_{R,i})$
13 $(st'_{R,i}, m) \leftarrow_\$ \mathsf{Recv}_i(st_{R,i}, ad, c)$
14 If $m = \bot$: $st \leftarrow \bot$
15 Else: $st \leftarrow (st_S, st'_{R,i})$
16 Return (st, m)

Figure 5.7: *Generic construction of a bidirectional channel* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *from unidirectional channels* $\mathsf{Ch}_1 = (\mathsf{Init}_1, \mathsf{Send}_1, \mathsf{Recv}_1)$ *and* $\mathsf{Ch}_2 = (\mathsf{Init}_2, \mathsf{Send}_2, \mathsf{Recv}_2)$*. We often refer to this construction as the 'canonic composition' of unidirectional channels.*

passive attacks on FIFO channels (F-IND-CPA). With other words, confidentiality against passive adversaries can be lifted from the unidirectional channels to the composed protocol, as stated in Theorem 8. Similarly, if both Ch_1 and Ch_2 offer plaintext integrity (INT-PTXT), respectively, ciphertext integrity (INT-CTXT), then the composed channel Ch provides the corresponding FIFO flavors of integrity (F-INT-PTXT and F-INT-CTXT, respectively); this is stated in Theorem 9.

Theorem 8 (Confidentiality Against Passive Adversaries)**.** *Let* Ch_1 *and* Ch_2 *be unidirectional channels and let* Ch *be the bidirectional channel obtained by composing them as described in Construction 3. If both* Ch_1 *and* Ch_2 *offer indistinguishability under chosen-plaintext attack, so does* Ch *as a FIFO channel. More precisely, for every adversary* $\mathcal{A}$ *attacking the F-IND-CPA property of* Ch *there exist adversaries* $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *against the IND-CPA property of* Ch_1 *and* Ch_2 *respectively, such that*

$$\mathbf{Adv}^{\text{F-IND-CPA}}_{\mathsf{Ch},2,\mathcal{A}}(\lambda) \leq \mathbf{Adv}^{\text{IND-CPA}}_{\mathsf{Ch}_1,\mathcal{B}_1}(\lambda) + \mathbf{Adv}^{\text{IND-CPA}}_{\mathsf{Ch}_2,\mathcal{B}_2}(\lambda) \ .$$

Proof. The idea is to define an intermediate game parametrized by two independent bits b and d (if $b = d$ we have exactly the original indistinguishability game). Now for each fixed value of d we can simulate the intermediate game using an IND-CPA adversary against one of the unidirectional channels. Formally, for $b \in \{0, 1\}$ let $\mathrm{E}^b_\mathcal{A}$ denote the F-IND-CPA game from Figure 5.2 involving an adversary $\mathcal{A}$ against Ch, and denote by $\Pr[\mathrm{E}^b_\mathcal{A}]$ the probability $\Pr[\mathrm{E}^b_\mathcal{A}(1^\lambda) = 1]$. We proceed by game hopping. The first game, that we denote by $\mathrm{E}^{0,0}_\mathcal{A}$, is the same as $\mathrm{E}^0_\mathcal{A}$. Define $\mathrm{E}^{0,1}_\mathcal{A}$ from $\mathrm{E}^{0,0}_\mathcal{A}$ by modifying the left-or-right oracle as follows: when a query (i, ad, m_0, m_1) is posed, invoke Send on message m_0 (as in the original game) if $i = 1$ and on message m_1 if $i = 2$. In other words, $\mathrm{E}^{0,1}_\mathcal{A}$ selects the 'left' message if the sender is Alice and the 'right' message if the sender is Bob. In the next hop, define $\mathrm{E}^{1,1}_\mathcal{A}$ from game $\mathrm{E}^{0,1}_\mathcal{A}$ by making the left-or-right oracle invoke Send always on message m_1. Note that $\mathrm{E}^{1,1}_\mathcal{A} = \mathrm{E}^1_\mathcal{A}$. We can bound $\mathcal{A}$'s advantage in the original game as follows:

$$\mathbf{Adv}^{\text{F-IND-CPA}}_{\mathsf{Ch},2,\mathcal{A}}(\lambda) \leq \left|\Pr[\mathrm{E}^{1,1}_\mathcal{A}] - \Pr[\mathrm{E}^{0,1}_\mathcal{A}]\right| + \left|\Pr[\mathrm{E}^{0,1}_\mathcal{A}] - \Pr[\mathrm{E}^{0,0}_\mathcal{A}]\right| \ .$$

We show next that the difference in probability between games $\mathrm{E}^{1,1}_{\mathcal{A}}$ and $\mathrm{E}^{0,1}_{\mathcal{A}}$, and between games $\mathrm{E}^{0,1}_{\mathcal{A}}$ and $\mathrm{E}^{0,0}_{\mathcal{A}}$, can be upper bounded by the IND-CPA advantage of efficient adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ against the unidirectional channels Ch_1 and Ch_2 respectively. Note that either of the above combinations of games fixes one of the two selection bits. For instance, both games $\mathrm{E}^{1,1}_{\mathcal{A}}$ and $\mathrm{E}^{0,1}_{\mathcal{A}}$ make Bob send the 'left' message. This combination of games implicitly defines a new indistinguishability game $\mathrm{E}^{b,1}_{\mathcal{A}}$—where $\mathcal{A}$ has to tell apart $\mathrm{E}^{1,1}_{\mathcal{A}}$ and $\mathrm{E}^{0,1}_{\mathcal{A}}$—for which we can predict $\mathcal{O}_{\mathsf{LoR}}$'s answers to queries $(i = 2, ad, m_0, m_1)$ by invoking Send_2 on message m_1. In fact, the latter observation is the basic working principle of the reduction $\mathcal{B}_1$ which runs $\mathcal{A}$ internally and answers $\mathcal{A}$'s queries using algorithm Send_2 and Recv_2 and the oracles provided by the IND-CPA game (defined in Figure 5.5) against channel Ch_1. A full description of the reduction $\mathcal{B}_1$ is given in Figure 5.8. It is immediate to see that $\mathcal{B}_1$ provides a perfect simulation of game $\mathrm{E}^{b,1}_{\mathcal{A}}$. To bound $\mathcal{A}$'s distinguishing advantage in game $\mathrm{E}^{b,1}_{\mathcal{A}}$ with $\mathcal{B}_1$'s advantage it suffices to show

---

$\mathcal{B}_1^{\mathcal{O}_{\mathsf{LoR}},\mathcal{O}^*_{\mathsf{Recv}}}(1^\lambda)$:
01 $G \leftarrow (\emptyset, \emptyset, \emptyset)$
02 $Q[\,] \leftarrow \emptyset$
03 $(st_S, st_R) \leftarrow_{\$} \mathsf{Init}_2(1^\lambda)$
04 $b' \leftarrow_{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{LoR}},\mathcal{O}^*_{\mathsf{Recv}}}(1^\lambda)$
05 Terminate with b'

If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{LoR}}(i, ad, m^0, m^1)$:
06 If $i = 1$:
07 $\quad d \leftarrow \mathcal{O}_{\mathsf{LoR}}(ad, m^0, m^1)$
08 Else:
09 $\quad (st_S, c) \leftarrow_{\$} \mathsf{Send}_2(st_S, ad, m^1)$
10 $G \leftarrow G + (\mathtt{S}, i)$ with v
11 $Q[v] \leftarrow (ad, c)$
12 Return c to $\mathcal{A}$

If $\mathcal{A}$ queries $\mathcal{O}^*_{\mathsf{Recv}}(i, j, ad, c)$:
13 $G \leftarrow G + (\mathtt{R}, i, j)$ with v
14 If $G = \perp \vee (ad, c) \neq Q[\omega(v)]$:
15 $\quad$ Terminate with 1
16 If $i = 1$:
17 $\quad \diamond \leftarrow_{\$} \mathcal{O}^*_{\mathsf{Recv}}(ad, c)$
18 Else:
19 $\quad (st_R, m) \leftarrow_{\$} \mathsf{Recv}_1(st_R, ad, c)$
20 $\quad$ Return $\diamond$ to $\mathcal{A}$

Figure 5.8: *Reduction $\mathcal{B}_1$ described in the proof for IND-CPA of Construction 3. $\mathcal{B}_1$ simulates the F-IND-CPA oracles in the right-to-left direction using Ch_2 and attacks the left-to-right direction (channel Ch_1). Similarly we can construct a specular reduction $\mathcal{B}_2$ which emulates the left-to-right direction and attacks the opposite.*

that if all of $\mathcal{A}$'s queries are passive, i.e., do not cause premature termination of game $\mathrm{E}^{b,1}_{\mathcal{A}}$, then the corresponding queries that $\mathcal{B}_1$ poses in the outer IND-CPA game are passive as well. Let $q = (i, j, ad, d)$ be any of $\mathcal{A}$'s receiving queries and suppose that q does not trigger the execution of instruction 13 (in Figure 5.2). If $i = 2$ (and $j = 1$) there is nothing to show: $\mathcal{B}_1$ answers the query on its own by invoking algorithm Recv_1 on input the current state st_R, associated data ad and ciphertext c. In the opposite case, i.e., $i = 1$ (and $j = 2$), $\mathcal{B}_1$ must ask query $q' = (ad, c)$ to $\mathcal{O}^*_{\mathsf{Recv}}$, which may force termination. As we show next, the IND-CPA game will not abort when $\mathcal{B}_1$ poses query q'. Let r_i and s_j denote the numbers of receiving actions performed by user i and sending actions performed by user j respectively. By the assumption of passiveness of $\mathcal{A}$ we know that operation '$G + (\mathtt{R}, 1, 2)$ with v' does not fail and $Q[\omega(v)] = (ad, c)$. This means that, when query (ad, c) is posed, Alice $(i = 1)$ can receive from Bob $(j = 2)$. With other words: $r_1 \leq s_2$ and the pair (ad, c) corresponds with the pair that Bob sent with his r_1-th invocation of Recv. The (negation of the) condition above can be immediately recognized in line 12 of the IND-CPA experiment (from Figure 5.5), hence $\mathcal{B}_1$ can safely ask query q' to $\mathcal{O}^*_{\mathsf{Recv}}$. We conclude that for each passive query q posed by $\mathcal{A}$ in the F-IND-CPA game the corresponding query q' that $\mathcal{B}_1$ poses in the IND-CPA game is passive as well. This allows us to bound $|\Pr[\mathrm{E}^{1,1}_{\mathcal{A}}] - \Pr[\mathrm{E}^{0,1}_{\mathcal{A}}]| \leq \mathbf{Adv}^{\text{IND-CPA}}_{\mathsf{Ch},\mathcal{B}_1}(\lambda)$. Using a similar strategy we can construct a reduction $\mathcal{B}_2$ which, symmetrically to $\mathcal{B}_1$, attacks the unidirectional channel Ch_2 and emulates game $\mathrm{E}^{0,b}_{\mathcal{A}}$ using the IND-CPA oracles and algorithms Send_1 and Recv_1. After deriving the second inequality $|\Pr[\mathrm{E}^{0,1}_{\mathcal{A}}] - \Pr[\mathrm{E}^{0,0}_{\mathcal{A}}]| \leq \mathbf{Adv}^{\text{IND-CPA}}_{\mathsf{Ch},\mathcal{B}_2}(\lambda)$ we obtain the desired bound for $\mathcal{A}$'s

advantage in the original game:

$$\mathbf{Adv}^{\text{F-IND-CPA}}_{\mathsf{Ch},2,\mathcal{A}}(\lambda) \leq \mathbf{Adv}^{\text{IND-CPA}}_{\mathsf{Ch}_1,\mathcal{B}_1}(\lambda) + \mathbf{Adv}^{\text{IND-CPA}}_{\mathsf{Ch}_2,\mathcal{B}_2}(\lambda) \ .$$

□

Theorem 9 (Integrity)**.** *Let* Ch_1 *and* Ch_2 *be two unidirectional channels, and let* Ch *be their canonic composition. If both* Ch_1 *and* Ch_2 *offer ciphertext integrity, then so does* Ch*. More precisely, for every efficient adversary* $\mathcal{A}$ *attacking the F-INT-CTXT property of* Ch *there exist efficient adversaries* $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *against the INT-CTXT property of* Ch_1 *and* Ch_2 *respectively, such that*

$$\mathbf{Adv}^{\text{F-IND-CPA}}_{\mathsf{Ch},2,\mathcal{A}}(\lambda) \leq \mathbf{Adv}^{\text{IND-CPA}}_{\mathsf{Ch}_1,\mathcal{B}_1}(\lambda) + \mathbf{Adv}^{\text{IND-CPA}}_{\mathsf{Ch}_2,\mathcal{B}_2}(\lambda) \ .$$

A similar statement holds for plaintext integrity, i.e., INT-PTXT of the unidirectional channels implies F-INT-PTXT of their canonic composition.

$\mathcal{B}_1^{\mathcal{O}_{\mathsf{Send}},\mathcal{O}_{\mathsf{Recv}}}(1^\lambda)$:
01 $G \leftarrow (\emptyset,\emptyset,\emptyset)$
02 $Q[\,] \leftarrow \emptyset$
03 $(st_S, st_R) \leftarrow_{\$} \mathsf{Init}_2(1^\lambda)$
04 $\mathcal{A}^{\mathcal{O}_{\mathsf{Send}},\mathcal{O}_{\mathsf{Recv}}}(1^\lambda)$
05 Terminate with 0

If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Send}}(i, ad, m)$:
06 If $i = 1$:
07 $\quad d \leftarrow \mathcal{O}_{\mathsf{Send}}(ad, m)$
08 Else:
09 $\quad (st_S, c) \leftarrow_{\$} \mathsf{Send}_2(st_S, ad, m)$
10 $G \leftarrow G + (\mathtt{S}, i)$ with v
11 $Q[v] \leftarrow (ad, c)$
12 Return c to $\mathcal{A}$

If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Recv}}(i, j, ad, c)$:
13 If $i = 2$:
14 $\quad m \leftarrow \mathcal{O}_{\mathsf{Recv}}(ad, c)$
15 $\quad$ If $m = \perp$:
16 $\quad\quad$ Return $\perp$ to $\mathcal{A}$
17 Else:
18 $\quad (st_R, m) \leftarrow_{\$} \mathsf{Recv}_2(st_R, ad, c)$
19 $\quad$ If $m = \perp$:
20 $\quad\quad$ Return $\perp$ to $\mathcal{A}$
21 $G \leftarrow G + (\mathtt{R}, i, j)$ with v
22 If $G = \perp \vee (ad, c) \neq Q[\omega(v)]$:
23 $\quad$ Terminate with 1
24 Else:
25 $\quad$ Return m to $\mathcal{A}$

Figure 5.9: *Reduction* $\mathcal{B}_1$ *described in the proof for INT-CTXT of Construction 3.* $\mathcal{B}_1$ *simulates the right-to-left direction using* Ch_2 *and attacks the left-to-right direction (channel* Ch_1*). A similar reduction* $\mathcal{B}_2$ *which emulates the left-to-right direction and attacks the opposite direction is easy to construct.*

Proof. For the proof we fix ATK = CTXT and refer to the F-INT-CTXT experiment from Figure 5.3 (on page 59), involving an adversary $\mathcal{A}$ against Ch, with the shortcut $\mathrm{E}^0_{\mathcal{A}}$. It is immediate to adapt the proof to the F-INT-PTXT case. Write $\Pr[\mathrm{E}^0_{\mathcal{A}}]$ to indicate the probability $\Pr[\mathrm{E}^0_{\mathcal{A}}(1^\lambda) = 1]$. Let bad_{LR} be the event that $\mathcal{A}$ breaks the F-INT-CTXT of Ch for $i = 2$ with a query (ad, d) to $\mathcal{O}_{\mathsf{Recv}}$. i.e., successful termination of the experiment is caused by a receiving query to Bob. Intuitively, in this case $\mathcal{A}$ breaks integrity when attacking the left-to-right direction (i.e., Alice sends and Bob receives). Analogously, denote by bad_{RL} the event that $\mathcal{A}$ breaks integrity when attacking the right-to-left direction, i.e., with a receiving query to Alice. We proceed by game-hopping: let us add the instruction 'if $i = 2$: Terminate with 0' after line 30 of game $\mathrm{E}^0_{\mathcal{A}}$ (defined in Figure 5.3), and denote the resulting game by $\mathrm{E}^1_{\mathcal{A}}$. The new game prevents $\mathcal{A}$ from attacking the left-to-right channel. Thus, games $\mathrm{E}^0_{\mathcal{A}}$ and $\mathrm{E}^1_{\mathcal{A}}$ execute the same instructions as long as event bad_{LR} does not occur. Similarly, define game $\mathrm{E}^2_{\mathcal{A}}$ as a modification of game $\mathrm{E}^1_{\mathcal{A}}$ by adding the instruction 'if $i = 1$: Terminate with 0' after the newly inserted line. In other words, game $\mathrm{E}^2_{\mathcal{A}}$ also prevents $\mathcal{A}$ from attacking the right-to-left direction. Since $\mathrm{E}^1_{\mathcal{A}}$ and $\mathrm{E}^2_{\mathcal{A}}$ differ only if event bad_{RL} occurs, we derive the inequalities

$|\Pr[\mathrm{E}^0_{\mathcal{A}}] - \Pr[\mathrm{E}^1_{\mathcal{A}}]| \leq \Pr[bad_{LR}]$ and $|\Pr[\mathrm{E}^1_{\mathcal{A}}] - \Pr[\mathrm{E}^2_{\mathcal{A}}]| \leq \Pr[bad_{RL}]$. Finally, since $\Pr[\mathrm{E}^2_{\mathcal{A}}] = 0$ we obtain the following bound for $\mathcal{A}$'s advantage in the original game:

$$\mathbf{Adv}^{\text{F-INT-CTXT}}_{\mathsf{Ch},2,\mathcal{A}} \leq \left|\Pr[\mathrm{E}^0_{\mathcal{A}}] - \Pr[\mathrm{E}^1_{\mathcal{A}}]\right| + \left|\Pr[\mathrm{E}^1_{\mathcal{A}}] - \Pr[\mathrm{E}^2_{\mathcal{A}}]\right| + \left|\Pr[\mathrm{E}^2_{\mathcal{A}}]\right| \leq \Pr[bad_{LR}] + \Pr[bad_{RL}] \ .$$

It remains to bound the probabilities of events bad_{LR} and bad_{RL}. To this end, we construct an adversary $\mathcal{B}_1$, which runs $\mathcal{A}$ internally, and show that $\mathcal{B}_1$ breaks integrity of Ch_1 as soon as $\mathcal{A}$ provokes event bad_{LR}. Briefly, $\mathcal{B}_1$ answers $\mathcal{A}$'s queries in the right-to-left direction using algorithms Init_2, Send_2 and Recv_2, while it forwards to its own oracles (provided by the INT-CTXT experiment against Ch_1) all queries affecting the opposite direction. A full specification of $\mathcal{B}_1$'s code is given in Figure 5.9. Note that bad_{LR} occurs during an execution of $\mathcal{B}_1$ if, for $i = 2$, $\mathcal{A}$ asks a query (ad, d) to $\mathcal{O}_{\mathsf{Recv}}$ which triggers the execution of instruction 23 (Figure 5.9). We claim that, in this case case, $\mathcal{A}$'s query also causes successful termination of the outer INT-CTXT game that $\mathcal{B}_1$ plays against Ch_1. Indeed, $\mathcal{A}$ triggers bad_{LR} if either the operation $G + (\mathtt{R}, 2, 1)$ is not admissible or the corresponding value of $Q[v]$ does not match the value of $Q[\omega(v)]$. By definition of the FIFO addition '+' we have that $G + (\mathtt{R}, 2, 1) = \bot$ if and only if Bob receives more often than Alice had send. The latter condition is precisely encoded as '$r > s$' (see line 32 of Figure 5.5) in the unidirectional INT-CTXT experiment. Similarly, having a mismatch between $Q[v]$ and $Q[\omega(v)]$ can be translated, in the unidirectional case, as '$ad \neq ad_r \vee c \neq c_r$' (see line 32 of Figure 5.5). Hence, causing event bad_{LR} directly implies an integrity breach of channel Ch_1, from which we derive the inequality $\Pr[bad_{LR}] \leq \mathbf{Adv}^{\text{INT-CTXT}}_{\mathsf{Ch}_1,\mathcal{B}_1}(\lambda)$. Using a similar argument in the opposite direction we obtain a reduction $\mathcal{B}_2$ against Ch_2 that wins the INT-CTXT game as soon as event bad_{RL} is triggered, and derive the final bound:

$$\mathbf{Adv}^{\text{F-INT-CTXT}}_{\mathsf{Ch},2,\mathcal{A}}(\lambda) \leq \mathbf{Adv}^{\text{INT-CTXT}}_{\mathsf{Ch}_1,\mathcal{B}_1}(\lambda) + \mathbf{Adv}^{\text{INT-CTXT}}_{\mathsf{Ch}_2,\mathcal{B}_2}(\lambda) \ .$$

□

We learned from Theorems 8 and 9 that the canonic composition of unidirectional channels preserves integrity of its unidirectional components as well as confidentiality against passive adversaries. However, as highlighted by the attack from Figure 5.1, confidentiality against active adversaries is not preserved. In the following we discuss why this is the case by revisiting the attack from a formal perspective.

Confidentiality against active adversaries is not preserved. We anticipated earlier in this chapter that indistinguishability against a chosen-ciphertext attack cannot be lifted from the unidirectional building blocks to their canonic composition generically. Here we revisit the attack from Figure 5.1 in light of our formalisms, showing a successful chosen-ciphertext attack (F-IND-CCA) against the canonic composition of two unidirectional channels which are both indistinguishable against chosen-ciphertext attacks (IND-CCA). Let $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ be the canonic composition of two unidirectional channels Ch_1 and Ch_2: The communication from Alice to Bob is protected by Ch_1 and the communication from Bob to Alice is protected by Ch_2. The claim is that if both Ch_1 and Ch_2 guarantee confidentiality against active attacks (more precisely: provide IND-CCA security), their canonic composition Ch in general does not. We prove the claim by restating the attack from Figure 5.1 in the language of Section 5.2. Let's recall the attack strategy: the adversary delivers a ciphertext to Alice that decrypts to 'please authenticate', obtains an encryption of Alice's password (which is so far protected because the unidirectional channels in use are confidential), and forwards this ciphertext to Bob who, upon receiving an unexpected string of random-looking characters, reacts by making this string (which, by correctness of the left-to-right channel, is precisely Alice's password) public.

Observe that the assumptions on Ch_1 and Ch_2 are exclusively on confidentiality and not on integrity, so we are free to assume that the corresponding Recv algorithms never reject.[7] In the following, we refer to Alice and Bob as participants 1 and 2, respectively, and ignore associated data for clarity.

Let $\mathcal{A}$ be an adversary that interacts in the F-IND-CCA game against Ch as follows. The adversary starts with choosing a ciphertext $\tilde{c}$ and two messages $m_0 \neq m_1$, submits a query $(1, 2, \tilde{c})$ to $\mathcal{O}_{\mathsf{Recv}}$ where she obtains a message $\tilde{m}$ (from Alice), a query $(1, m_0, m_1)$ to $\mathcal{O}_{\mathsf{LoR}}$ that produces a ciphertext c (from Alice), and a query $(2, 1, c)$ to $\mathcal{O}_{\mathsf{Recv}}$ where she obtains a message m' or $\diamond$ (from Bob), as we see next. By the rules of the F-IND-CCA experiment (Figure 5.2) the first query is identified as active (Alice receives although nothing has been sent by Bob; formally, $G = (\emptyset, \emptyset, \emptyset)$ and hence the graph operation $G + (\mathtt{R}, 1, 2)$ is invalid) and, thus, $\mathsf{active}_1 \leftarrow 1$ is set in line 31. The third query is also identified as active (because $G = \bot$, see line 12) and the oracle returns c's decryption $m' = m_b$ to $\mathcal{A}$. The adversary outputs 1 iff $m' = m_1$. It is easy to check that its advantage is 1.

Note that the attack crucially involves two directions of communication and thus cannot be expressed within the security models of Bellare *et al.* [BKN02] and its numerous refinements [Nam02, KPB03, BDPS12, JKSS12, BDPS14, KPW13, BSWW13, FGMP15], as all these models restrict the attention to the case of one direction of communication. While the attack confirms our approach to model bidirectional channels *as a whole*—in contrast to only assessing the security of their unidirectional components—we point out that it does not apply to many real-world protocols (including TLS) that, besides confidentiality, also provide integrity protection. The reason is simple: Any meaningful integrity notion would make Alice reject ciphertext $\tilde{c}$, thus her sending operation would never take place.

Security of the canonic composition. While highlighting the importance of analyzing the exact security guarantees that a combined cryptographic scheme inherits from its building blocks, we stress that the attack from Figure 5.1 does not directly translate to actual weaknesses of real-world bidirectional channels, like TLS and SSH, that follow the canonic composition design. In fact, these protocols combine two unidirectional channels that offer, beyond confidentiality, also strong integrity guarantees. Established results in [BKN02] prove some variants of the SSH channel to be secure the stateful AE sense; [PRS11] show that the CBC-based TLS record protocol, version 1.1 and 1.2, provably meets (a stronger variant of) stateful AEAD security. In our terminology: the unidirectional components of both channel protocols offer INT-CTXT and IND-CPA security. Using the result of Theorem 5 we conclude that SSH and TLS provably achieve FIFO confidentiality against active adversaries (F-IND-CCA) as well as integrity of ciphertexts (F-INT-CTXT), validating the soundness of their design as bidirectional channels.

Corollary 1 (Security of the canonic composition)**.** *Let* Ch_1 *and* Ch_2 *be two unidirectional channels, and let* Ch *be their canonic composition. If both* Ch_1 *and* Ch_2 *offer ciphertext integrity and indistinguishability under chosen-message attacks, then* Ch *offers the strongest notions of integrity and confidentiality for FIFO channels. More precisely, if* Ch_1 *and* Ch_2 *are INT-CTXT and IND-CPA-secure then* Ch *is F-INT-CTXT- and F-IND-CCA-secure.*

[7] Any confidential channel where Recv potentially rejects can be turned into a channel where Recv never rejects by modifying it such that instead of rejecting it restricts itself to outputting some fixed message m_0. This modification does not affect the confidentiality properties of the channel. (Its integrity properties might very well be affected, but this does not matter in the current context.)

Chapter 6

Causal Channels

In this chapter we introduce causal channels, which essentially provide a secure version of causal broadcast protocols. Beyond preserving causality, these channels offer an extended functionality which lets users obtain the current communication history upon sending and receiving.

6.1 Introduction

Consider a chatroom situation like the one illustrated in Figure 6.1. Alice starts a conversation by asking around what the other participants would never do in their life. Bob would never withdraw his current submission to HESSECRYPT and answers by writing 'withdraw the submission!'. Charlie, a co-author of Bob, missed Alice's initial question (possibly by arrangement of the adversary), by consequence misunderstands Bob, and ultimately withdraws their joint paper against Bob's will.

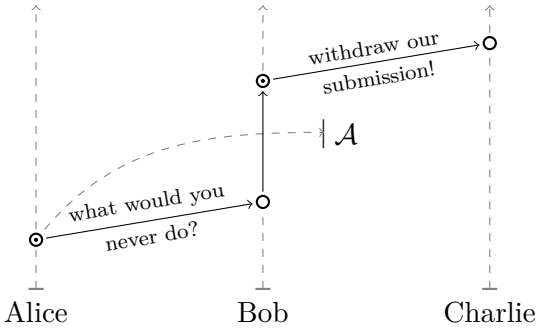


Figure 6.1: *A misunderstanding caused by a violation of the causal property (CAUS). Vertical dashed lines symbolize per-party timelines, that sending actions are marked with ⊙ and receiving actions with ○.*

A standard result in the domain of distributed computing is that the technical solution to misunderstandings of this type is to deploy a specialized broadcast protocol that preserves *causality*, i.e., that always delivers messages according to the logical order of their corresponding sending events: no participant shall receive a message if it did not also receive all messages that were sent before (here, 'before' is meant in a global sense, in contrast to FIFO ordering that specifies how messages sent by each specific sender should be delivered).[1] In the context of Figure 6.1, a causal broadcast protocol would ensure that Charlie would see Alice's message before seeing the one from Bob.

[1]The notion of time considered in causality is abstract and defined independently of physical time.

Protocols that efficiently realize a causal broadcast infrastructure are well-known in the distributed computing literature (some examples can be found in [AW04, CGR11]). However, such protocols typically do not give guarantees on the causal delivery of messages in the face of active adversaries tampering with the message deliveries. To cure this situation we extend our study to cryptographic broadcast channels and propose a new integrity notion, *causal integrity*, that considers the causal relationship between sending and receiving events. Beyond providing standard protection against message manipulation, this notion ensures that no adversary can tamper with the causal ordering of messages without being detected. If the chatroom from Figure 6.1 was implemented using a corresponding causal channel, the described misunderstanding would not have happened.

Conversation history. Consider next the situation suggested in Figure 6.2a. Alice, a stockholder, and Bob, her financial adviser, use an online chat to discuss Alice's investment strategy. She asks: 'Shall I sell my shares?', but Bob remains silent. Alice understands he is currently not available. A bit later she poses a different question: 'Shall I increase my investment?' This time, she gets an answer from Bob: 'Definitely, the sooner the better!' Alice follows Bob's advice and purchases stocks. By mistake: In reality, as illustrated in Figure 6.2b, the adversary created a misunderstanding by delaying the network transmission from Bob to Alice. Alice buys, but Bob actually proposed to sell.

(a) *Alice's view* **(b)** *The reality*

Figure 6.2: *An attack exploiting that Alice does not see the communication history. Vertical dashed lines symbolize per-party timelines, that sending actions are marked with ⊙ and receiving actions with ○.*

Observe that the described problem does not stem from an alteration of the causality of events: all messages are indeed delivered consistently following the causal order in which they were sent. Rather, the misunderstanding arises because Alice learns from her channel only the content of the received messages, but not how these messages are causally related to the messages she sent. In particular, she does not get explicit information about which incoming message is a response to which outgoing message (for instance, that Bob's answer was 'caused' by Alice's first message, not by her second message). This is different if the causal channel is *history-reporting*, i.e., if it informs the participants on every send and receive invocation about the past communication history.

In the present chapter we introduce this augmented type of channel, that we call a *causal channel*, and study the security that we expect from it. In particular, beyond confidentiality and integrity, we introduce the new security notion of *history integrity*, demanding that the communication history is always reported correctly. If Alice and Bob had deployed a causal channel that provides history integrity they wouldn't have had the misunderstanding described

above. Rather, Alice would have learned from the incoming message that the latter is an answer to her first question, not to the second.

6.2 Syntax and Functionality

Syntactically, a causal channel differs from a FIFO channel in that Send and Recv algorithms return an auxiliary output h, called *history*. We denote this also by saying that the channel is *history reporting*. Intuitively, the history gives an overview of the sending and receiving actions that are 'acknowledgeable' by the user. With other words, the history obtained when executing a sending or receiving action shall inform the user that performed that action of all actions that causally precede her/his last action.

Definition 25 (Syntax of causal channels)**.** *A* causal channel *with associated data space $\mathcal{AD}$, message space $\mathcal{M}$, ciphertext space $\mathcal{C}$, and state space $\mathcal{S}$ is a tuple* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *of efficient algorithms as follows:*

- Init*. The initialization algorithm takes as input a security parameter 1^λ and an integer N, and outputs initial states $st_1, \ldots, st_N \in \mathcal{S}$. We write $(st_1, \ldots, st_N) \leftarrow_\$ \mathsf{Init}(1^\lambda, N)$.*
- Send*. The sending algorithm takes as input a state $st \in \mathcal{S}$, associated data $ad \in \mathcal{AD}$, and a message $m \in \mathcal{M}$, and outputs a state $st' \in \mathcal{S}$, a ciphertext $c \in \mathcal{C}$ or $c = \perp$, and a history $h \in \mathcal{G}$. We write $(st', c, h) \leftarrow_\$ \mathsf{Send}(st, ad, m)$.*
- Recv*. The receiving algorithm takes as input a state $st \in \mathcal{S}$, an origin indicator $j \in [1\,..\,N]$, associated data $ad \in \mathcal{AD}$, and a ciphertext $c \in \mathcal{C}$, and outputs a state $st' \in \mathcal{S}$, a message $m \in \mathcal{M}$, and a history $h \in \mathcal{G}$, or it outputs $st' = m = h = \perp$. We write $(st', m, h) \leftarrow_\$ \mathsf{Recv}(st, j, ad, c)$, correspondingly.*

For causal channels, beyond the standard correct message recovery we also require that the history be reported correctly, as the next definition specifies. Formally, given a causal graph $G = (V, \leq, \chi)$ the history of a given action $v \in V$ is represented by $G^{(v)}$, the v-prefix of G (see Definition 15 on page 46), which is also a causal graph by Lemma 3 (on page 48).

Definition 26 (Correctness of causal channels)**.** *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a causal channel with associated data space $\mathcal{AD}$, message space $\mathcal{M}$, ciphertext space $\mathcal{C}$, and state space $\mathcal{S}$. Let $G = (V, \leq, \chi)$ be an N-party communication graph and for $n = |V|$ assume an enumeration $e\colon [1\,..\,n] \to V$ of $(V, \leq)$. Let $\alpha\colon V^S \to \mathcal{AD}$ and $\mu\colon V^S \to \mathcal{M}$ denote arbitrary assignments. Denote by $c[\,]$, $m[\,]$ and $h[\,]$ associative arrays that map $V^S \to \mathcal{C}$, $V^R \to \mathcal{M}$, and $V \to \mathcal{G}$ respectively. Consider the following procedure:*

---

01 Initialize states $(st_1, \ldots, st_N) \leftarrow_\$ \mathsf{Init}(1^\lambda, N)$
02 Process the actions $v \in V$ in order $v_1 = e(1), \ldots, v_n = e(n)$ according to the rules:
03 – if $\chi(v) = (\mathtt{S}, i)$ then $(st_i, c[v], h[v]) \leftarrow_\$ \mathsf{Send}(st_i, \alpha(v), \mu(v))$
04 – if $\chi(v) = (\mathtt{R}, i, j)$ then $(st_i, m[v], h[v]) \leftarrow_\$ \mathsf{Recv}(st_i, j, \alpha(\omega(v)), c[\omega(v)])$

We say that a causal channel Ch *is correct if for every causal graph $G \in \mathcal{G}_{\text{CAUS}}$, for all e, α, μ, and for all choices of the randomness of the* Init*,* Send*, and* Recv *algorithms, the* Recv *algorithm in the procedure correctly recovers all sent messages, i.e., if $m[v] = \mu(\omega(v))\,\forall v \in V^R$ and, in addition, the history output is accurate, i.e., if $h[v] = G^{(v)}$ for all $v \in V$.*

6.3 Security and Relations Among Notions

Similarly to the case of FIFO channels (Chapter 5), we model confidentiality and integrity of causal channels via experiments in which an adversary $\mathcal{A}$ interacts with the Send and Recv algorithms through oracles. In Figures 6.3 and 6.4 we specify the indistinguishability experiments C-IND-CPA and C-IND-CCA, respectively, the integrity experiments C-INT-PTXT and C-INT-CTXT for causal channels. Security is defined analogously to the FIFO setting. The main difference in the causal setting is concentrated in the condition that declares adversarial queries as 'active'. Here, this condition is triggered if a ciphertext manipulation occurs or if the communication scheduled by the adversary violates the causal ordering property. In the experiments, the latter is tested by letting the challenger maintain a causal graph G to record (parts of) the actions scheduled by the adversary.

Syntactically, the security experiments for causal channels differ from the corresponding FIFO experiments from Section 5.3 (on page 56) 'only' in that (i) Send and Recv produce an auxiliary output h, and (ii) the communication graph G which registers the scheduled communication shall be a causal graph rather than a FIFO graph; the latter is ensured by adding new actions to the graph via the $\oplus$ operation (if possible according to the addition rules), rather than the $+$ operation. We stress that, although (ii) optically results in a marginal change of the FIFO experiments, it has in fact a major effect on the winning conditions. Indeed, according to the causal experiments, an adversary that interferes with the deliveries such that the FIFO ordering holds but the causal ordering does not is also deemed as active. With other words, an attack (on confidentiality or on integrity) may be discarded in the FIFO setting while being considered successful in the causal setting, meaning that causal security provides strictly stronger guarantees than FIFO security.

History integrity. Exclusively for causal channels we also introduce a third integrity notion, *history integrity* (C-INT-HIST), demanding that no adversary be able to make any participant output a history h that is not in agreement with the actual communication history. That is, for an arbitrary sequence of $\mathcal{O}_{\mathsf{Send}}$ and $\mathcal{O}_{\mathsf{Recv}}$ operations the notion ensures that every participant after each query has the correct view on the communication structure that occurred up to that point. Technically this is expressed by the $h = G^{(i)}$ conditions in lines 37 and 43.

We illustrate the indistinguishability in Figure 6.3 and the integrity experiments in Figure 6.4. Security is defined in the usual way.

Definition 27 (Indistinguishability for causal channels)**.** *For* ATK $\in \{\text{CPA}, \text{CCA}\}$ *we say that a causal channel* Ch *offers* ATK-indistinguishability *if for all efficient adversaries* $\mathcal{A}$ *and all polynomial* $N = N(\lambda)$ *the following advantage function is negligible,*

$$\mathbf{Adv}^{\text{C-IND-ATK}}_{\mathsf{Ch},N,\mathcal{A}}(\lambda) = \left|\Pr\left[\text{Expt}^{\text{C-IND-ATK},1}_{\mathsf{Ch},N,\mathcal{A}}(1^\lambda) = 1\right] - \Pr\left[\text{Expt}^{\text{C-IND-ATK},0}_{\mathsf{Ch},N,\mathcal{A}}(1^\lambda) = 1\right]\right| \ .$$

*We abbreviate indistinguishability under chosen-plaintext attacks (*CPA*-indistinguishability) and indistinguishability under a chosen-ciphertext attacks (*CCA*-indistinguishability) for causal channels by writing C-IND-CPA and C-IND-CCA, respectively.*

Definition 28 (Integrity for causal channels)**.** *For* ATK $\in \{\text{PTXT}, \text{CTXT}, \text{HIST}\}$ *we say that a causal channel* Ch *offers* ATK-integrity *if for all efficient adversaries* $\mathcal{A}$ *and all polynomial* $N = N(\lambda)$ *the following advantage function is negligible,*

$$\mathbf{Adv}^{\text{C-INT-ATK}}_{\mathsf{Ch},N,\mathcal{A}}(\lambda) = \left|\Pr\left[\text{Expt}^{\text{C-INT-ATK}}_{\mathsf{Ch},N,\mathcal{A}}(1^\lambda) = 1\right]\right| \ .$$

$\mathrm{Expt}^{\text{C-IND-CPA},b}_{\mathsf{Ch},N,\mathcal{A}}(1^\lambda)$:
01 $G \leftarrow (\emptyset,\emptyset,\emptyset)$
02 $Q[\,] \leftarrow \emptyset$
03 $(st_1,\ldots,st_N) \leftarrow_\$ \mathsf{Init}(1^\lambda, N)$
04 $b' \leftarrow_\$ \mathcal{A}^{\mathcal{O}_{\mathsf{LoR}},\mathcal{O}^*_{\mathsf{Recv}}}(1^\lambda)$
05 Terminate with b'

$\mathcal{O}_{\mathsf{LoR}}(i, ad, m_0, m_1)$:
06 Require $|m_0| = |m_1|$
07 $(st_i, c, h) \leftarrow_\$ \mathsf{Send}(st_i, ad, m_b)$
08 $G \leftarrow G \oplus (\mathtt{S}, i)$ with v
09 $Q[v] \leftarrow (ad, c)$
10 Return (c, h) to $\mathcal{A}$

$\mathcal{O}^*_{\mathsf{Recv}}(i, j, ad, c)$:
11 $G \leftarrow G \oplus (\mathtt{R}, i, j)$ with v
12 If $G = \bot \vee (ad, c) \neq Q[\omega(v)]$:
13 Terminate with 0
14 Else:
15 $(st_i, m, h) \leftarrow_\$ \mathsf{Recv}(st_i, j, ad, c)$
16 Return $(\diamond, h)$ to $\mathcal{A}$

$\mathrm{Expt}^{\text{C-IND-CCA},b}_{\mathsf{Ch},N,\mathcal{A}}(1^\lambda)$:
17 $G \leftarrow (\emptyset,\emptyset,\emptyset)$
18 $Q[\,] \leftarrow \emptyset$
19 $\mathsf{active}_1 \leftarrow \cdots \leftarrow \mathsf{active}_N \leftarrow 0$
20 $(st_1,\ldots,st_N) \leftarrow_\$ \mathsf{Init}(1^\lambda, N)$
21 $b' \leftarrow_\$ \mathcal{A}^{\mathcal{O}_{\mathsf{LoR}},\mathcal{O}^*_{\mathsf{Recv}}}(1^\lambda)$
22 Terminate with b'

$\mathcal{O}_{\mathsf{LoR}}(i, ad, m_0, m_1)$:
23 Require $|m_0| = |m_1|$
24 $(st_i, c, h) \leftarrow_\$ \mathsf{Send}(st_i, ad, m_b)$
25 If $\mathsf{active}_i = 0$:
26 $G \leftarrow G \oplus (\mathtt{S}, i)$ with v
27 $Q[v] \leftarrow (ad, c)$
28 Return (c, h) to $\mathcal{A}$

$\mathcal{O}^*_{\mathsf{Recv}}(i, j, ad, c)$:
29 $G' \leftarrow G \oplus (\mathtt{R}, i, j)$ with v
30 If $G' = \bot \vee (ad, d) \neq Q[\omega(v)]$:
31 $\mathsf{active}_i \leftarrow 1$
32 $(st_i, m, h) \leftarrow_\$ \mathsf{Recv}(st_i, j, ad, c)$
33 If $\mathsf{active}_i = 1$:
34 Return m to $\mathcal{A}$
35 Else:
36 $G \leftarrow G'$
37 Return $\diamond$ to $\mathcal{A}$

Figure 6.3: *Indistinguishability experiments for causal channels. We assume that once a query results in state st_i being set to $\bot$, then no further queries for that participant are accepted; this is without loss of generality, as the channel algorithms would always reject for that participant. We further assume $(i, j) \in [\![1\,..\,N]\!]$, $ad \in \mathcal{AD}$, $m_0, m_1 \in \mathcal{M}$, and $c \in \mathcal{C}$ for all such values provided by the adversary. When writing '$G \oplus x$ with v' we use v as a placeholder for the node that is newly added to the causal graph G in case the operation $G \oplus x$ does not fail. We use sending actions as indices for the associative array $Q[\,]$. A flag active_i per participant is kept to register the participants that are affected by an active measure of the adversary (in which case $\mathsf{active}_i = 1$).*

*We abbreviate the notions of integrity of plaintexts (*PTXT*-integrity), integrity of ciphertexts (*CTXT*-integrity), and integrity of history (*HIST*-integrity) for causal channels by writing C-INT-PTXT, C-INT-CTXT, and C-INT-HIST, respectively.*

Relations among notions. Similar implications to those discussed in the context of FIFO channels also hold for causal channels, namely, C-IND-CCA $\Longrightarrow$ C-IND-CPA and C-INT-CTXT $\Longrightarrow$ C-INT-PTXT. This is in line with corresponding results for stateless encryption [BDJR97] and stateful encryption [BKN02]. Further, we have that ciphertext integrity implies history integrity, i.e., C-INT-CTXT $\Longrightarrow$ C-INT-HIST. To see why the latter relation holds, observe that C-INT-CTXT guarantees that all ciphertexts accepted by Recv were genuinely produced by $\mathcal{O}_{\mathsf{Send}}$ and are submitted for decryption according to the causal property, and thus they obey the correctness regime. Now, as the integrity experiments only record in G the actions that accepted by the channel algorithms (to ensure that G is a causal graph), it directly follows by correctness (Definition 26) that, for these actions, the history is accurate. In contrast, message integrity does not suffice: C-INT-PTXT $\not\Longrightarrow$ C-INT-HIST.[2] Thus, ciphertext integrity is the strongest of the integrity notions, both in the FIFO and in the causal case.

[2]Starting from a causal channel that is C-INT-HIST, construct a second causal channel from it by appending a redundant zero-bit to each ciphertext. Let Recv output an arbitrary history when being presented a ciphertext where this bit has value one.

$\mathrm{Expt}^{\text{C-INT-PTXT}}_{\mathsf{Ch},N,\mathcal{A}}(1^\lambda)$:
01 $G \leftarrow (\emptyset,\emptyset,\emptyset)$
02 $Q[\,] \leftarrow \emptyset$
03 $(st_1,\ldots,st_N) \leftarrow_\$ \mathsf{Init}(1^\lambda, N)$
04 $\mathcal{A}^{\mathcal{O}_{\mathsf{Send}},\mathcal{O}_{\mathsf{Recv}}}(1^\lambda)$
05 Terminate with 0

$\mathcal{O}_{\mathsf{Send}}(i, ad, m)$:
06 $(st_i, c, h) \leftarrow_\$ \mathsf{Send}(st_i, ad, m)$
07 $G \leftarrow G \oplus (\mathtt{S}, i)$ with v
08 $Q[v] \leftarrow (ad, m)$
09 Return (c, h) to $\mathcal{A}$

$\mathcal{O}_{\mathsf{Recv}}(i, j, ad, c)$:
10 $(st_i, m, h) \leftarrow_\$ \mathsf{Recv}(st_i, j, ad, c)$
11 If $st_i = \bot$: Return $\bot$ to $\mathcal{A}$
12 $G \leftarrow G \oplus (\mathtt{R}, i, j)$ with v
13 If $G = \bot \vee (ad, m) \neq Q[\omega(v)]$:
14 Terminate with 1
15 Return (m, h) to $\mathcal{A}$

$\mathrm{Expt}^{\text{C-INT-CTXT}}_{\mathsf{Ch},N,\mathcal{A}}(1^\lambda)$:
16 $G \leftarrow (\emptyset,\emptyset,\emptyset)$
17 $Q[\,] \leftarrow \emptyset$
18 $(st_1,\ldots,st_N) \leftarrow_\$ \mathsf{Init}(1^\lambda, N)$
19 $\mathcal{A}^{\mathcal{O}_{\mathsf{Send}},\mathcal{O}_{\mathsf{Recv}}}(1^\lambda)$
20 Terminate with 0

$\mathcal{O}_{\mathsf{Send}}(i, ad, m)$:
21 $(st_i, c, h) \leftarrow_\$ \mathsf{Send}(st_i, ad, m)$
22 $G \leftarrow G \oplus (\mathtt{S}, i)$ with v
23 $Q[v] \leftarrow (ad, c)$
24 Return (c, h) to $\mathcal{A}$

$\mathcal{O}_{\mathsf{Recv}}(i, j, ad, c)$:
25 $(st_i, m, h) \leftarrow_\$ \mathsf{Recv}(st_i, j, ad, c)$
26 If $st_i = \bot$: Return $\bot$ to $\mathcal{A}$
27 $G \leftarrow G \oplus (\mathtt{R}, i, j)$ with v
28 If $G = \bot \vee (ad, c) \neq Q[\omega(v)]$:
29 Terminate with 1
30 Return (m, h) to $\mathcal{A}$

$\mathrm{Expt}^{\text{C-INT-HIST}}_{\mathsf{Ch},N,\mathcal{A}}(1^\lambda)$:
31 $G \leftarrow (\emptyset,\emptyset,\emptyset)$
32 $(st_1,\ldots,st_N) \leftarrow_\$ \mathsf{Init}(1^\lambda, N)$
33 $\mathcal{A}^{\mathcal{O}_{\mathsf{Send}},\mathcal{O}_{\mathsf{Recv}}}(1^\lambda)$
34 Terminate with 0

$\mathcal{O}_{\mathsf{Send}}(i, ad, m)$:
35 $(st_i, c, h) \leftarrow_\$ \mathsf{Send}(st_i, ad, m)$
36 $G \leftarrow G \oplus (\mathtt{S}, i)$
37 If $G = \bot \vee h \neq G^{(i)}$:
38 Terminate with 1
39 Return (c, h) to $\mathcal{A}$

$\mathcal{O}_{\mathsf{Recv}}(i, j, ad, c)$:
40 $(st_i, m, h) \leftarrow_\$ \mathsf{Recv}(st_i, j, ad, c)$
41 If $st_i = \bot$: Return $\bot$ to $\mathcal{A}$
42 $G \leftarrow G \oplus (\mathtt{R}, i, j)$
43 If $G = \bot \vee h \neq G^{(i)}$:
44 Terminate with 1
45 Return (c, h) to $\mathcal{A}$

Figure 6.4: *Integrity experiments for causal channels. We assume that once a query results in state st_i being set to $\bot$, then no further queries for that participant are accepted. We further assume $(i, j) \in [\![1\,..\,N]\!]$, $ad \in \mathcal{AD}$, $m \in \mathcal{M}$, and $c \in \mathcal{C}$ for all such values provided by the adversary. When writing '$G \oplus x$ with v' we use v as a placeholder for the node that is newly added to the causal graph G in case the operation $G \oplus x$ does not fail. We use sending actions as indices for the associative array $Q[\,]$. A flag active_i per participant is kept to register the participants that are affected by an active measure of the adversary (in which case $\mathsf{active}_i = 1$).*

6.4 Constructions

Network infrastructures that add causality guarantees to the communication of many participants play an important role in distributed computing. Standard constructions are known that achieve such properties, but without giving any cryptographic guarantees. In Section 6.2 we introduced the corresponding cryptographic primitive, the causal channel, and proposed suitable security notions. Here we answer the remaining question on how to securely construct such a channel from simpler (cryptographic) building blocks.

We propose a construction of a secure causal channel that supports an arbitrary number of participants. It runs on top of a causal network infrastructure and uses a secure FIFO channel as a building block. We give rationale on our construction and formal security statements below. In a nutshell: if the FIFO channel provides corresponding security properties, the constructed causal channel meets all properties of integrity and confidentiality defined in the earlier sections of this thesis.

Construction 4 (Causal channels from FIFO channels)**.** *For an associated data space $\mathcal{AD}$, let $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ be a FIFO channel with associated data space $\mathcal{AD} \times \mathbb{N}^*$, message space $\mathcal{M}$, ciphertext space $\mathcal{C}$, and state space $\mathcal{S}$. The causal channel $\mathsf{Ch}_c = (\mathsf{Init}_c, \mathsf{Send}_c, \mathsf{Recv}_c)$*

$\mathsf{Init}_c(1^\lambda, N)$:
01 $(\hat{st}_1, \ldots, \hat{st}_N) \leftarrow_\$ \mathsf{Init}(1^\lambda, N)$
02 For $i \leftarrow 1$ to N:
03 $\quad I_i \leftarrow \epsilon$
04 $\quad h_i \leftarrow (\emptyset, \emptyset, \emptyset)$
05 $\quad st_i \leftarrow (\hat{st}_i, I_i, h_i)$
06 Return $(st_1, \ldots, st_N)$

$\mathsf{Send}_c(st_i, ad, m)$:
07 If $st_i = \perp$: Return $(\perp, \perp, \perp)$
08 Parse st_i as $(\hat{st}_i, I_i, h_i)$
09 $ad' \leftarrow (ad, I_i)$
10 $(\hat{st}_i, c') \leftarrow_\$ \mathsf{Send}(\hat{st}_i, ad', m)$
11 $c \leftarrow (I_i, c')$
12 $I_i \leftarrow \epsilon$
13 $h_i \leftarrow h_i \oplus (\mathtt{S}, i)$
14 $st_i \leftarrow (\hat{st}_i, I_i, h_i)$
15 Return (st_i, c, h_i)

$\mathsf{Recv}_c(st_i, j, ad, c)$:
16 Parse st_i as $(\hat{st}_i, I_i, h_i)$
17 If $st_i = \perp$: Return $(\perp, \perp, \perp)$
18 Parse c as $(\iota^1 \| \ldots \| \iota^t, c')$
19 If parsing fails:
20 $\quad st_i \leftarrow \perp$
21 $\quad$ Return $(\perp, \perp, \perp)$
22 $ad' \leftarrow (ad, \iota^1 \| \ldots \| \iota^t)$
23 $(\hat{st}_i, m) \leftarrow_\$ \mathsf{Recv}(\hat{st}_i, j, ad', c')$
24 If $\hat{st}_i = \perp$:
25 $\quad st_i \leftarrow \perp$
26 $\quad$ Return $(\perp, \perp, \perp)$
27 $I_i \leftarrow I_i \| j$
28 For $k \leftarrow 1$ to t:
29 $\quad h_i \leftarrow h_i \oplus (\mathtt{R}, j, \iota^k)$
30 $h_i \leftarrow h_i \oplus (\mathtt{S}, j) \oplus (\mathtt{R}, i, j)$
31 If $h_i = \perp$:
32 $\quad st_i \leftarrow \perp$
33 $\quad$ Return $(\perp, \perp, \perp)$
34 $st_i \leftarrow (\hat{st}_i, I_i, h_i)$
35 Return (st_i, m, h_i)

Figure 6.5: *Construction of causal channel* Ch_c *from a FIFO channel* Ch.

with associated data space $\mathcal{AD}$*, message space* $\mathcal{M}$*, ciphertext space* $\mathbb{N}^* \times \mathcal{C}$*, and state space* $\mathcal{S} \times \mathbb{N}^* \times \mathcal{G}$ *is specified in Figure 6.5.*

The working principle of our construction Ch_c is as follows. For the communication between peers it relies on a causal network and, on top of it, a FIFO channel Ch. The state variables st of the constructed Ch_c consist of the state $\hat{st}$ of the Ch instance, a string I which records the origin of received ciphertexts (line 27 of Recv_c) and is reset at each sending operation (line 12 of Send_c), and an internal communication graph h. As the communication proceeds, h is gradually extended by new actions such that it always reflects the participant's prefix of the ongoing conversation. Consequently, for sending operations only the current action needs to be recorded (line 13 of Send_c). For receiving actions the situation is more involved, as an excerpt of the communication history of the remote party has to be incorporated in the local representation (lines 28–30 of Recv_c). To do so we let participants include partial information about their communication history in transmitted ciphertexts (line 11 of Send_c) and authenticate it using the associated data field of Ch (line 09).

Remark 5 (Optimization in the two-party case). We point out that Construction 4 can be slightly optimized when employed in the two-party setting.[3] The crucial observation is that in this case the I component of the state variable effectively just counts the number of received ciphertexts. It could thus be replaced by a simpler variable to save memory and bandwidth.

We prove next that Construction 4 is correct (as a causal channel), if the underlying FIFO channel Ch is. This result is not as easy to show as it might first seem: for causal channels

[3] In the two-party setting FIFO security notions and causal security notions coincide (Lemma 2 on page 48), so the reason to use our construction would be to obtain the history output.

the correctness requirement goes beyond the standard message recovery but also demands the history output h to be consistent with the causal order of the sending and receiving events.

In the following three theorems and their corollary we denote with Ch_c the causal channel obtained by applying Construction 4 to a FIFO channel Ch.

Theorem 10 (Correctness of Construction 4)**.** *If* Ch *is a correct FIFO channel then* Ch_c *is a correct causal channel.*

Proof. Let $G = (V, \leq, \chi)$ be a causal graph and let $n = |V|$. Fix an enumeration $e\colon [1\,..\,n] \to V$ of $(V, \leq)$ and write $v_1 = e(1), \ldots, v_n = e(n)$. Finally, let $\alpha\colon V^S \to \mathcal{AD}$ and $\mu\colon V^S \to \mathcal{M}$ be arbitrary assignments. Proceed with performing the actions of V as specified in Definition 26 (on page 75), and denote with $G_0, \ldots, G_n$ the implicitly defined sequence of causal graphs such that $G_0 = (\emptyset, \emptyset, \emptyset)$ and $G_k = G_{k-1} \oplus \chi(v_k)$ for $1 \leq k \leq n$. Note that for every k we have $G^{(v_k)} = G_k^{(i)}$, where i indicates the participants that performs v_k, i.e., $v_k \in V_i$. We prove by induction that $v_k \in V^R \Longrightarrow m[v_k] = \mu(\omega(v_k))$ and $v_k \in V \Longrightarrow h[v_k] = G_k^{(i)}$, for the corresponding i. Let hence k be arbitrary and assume that the statement holds for all $k' < k$. Let $(i, j) \in [\![1\,..\,N]\!]$ be such that either $v_k \in V_i^S$ or $v_k \in V_{ij}^R$.

Consider first the case $v_k \in V_i^S$, i.e., $\chi(v_k) = (\mathtt{S}, i)$. If v_k is the smallest (i.e., first) action in V_i, then $h[v_k] = (\emptyset, \emptyset, \emptyset) \oplus (\mathtt{S}, i) = G_k^{(i)}$ is clear. Otherwise let v_l be the (unique) direct predecessor of v_k in V_i. Then $l < k$ and for all $l < r < k$ we have $v_r \notin V_i$. By the inductive assumption we have $h[v_l] = G_l^{(i)}$. By applying Lemma 8(b) sufficiently often we obtain

$$h[v_k] = h[v_l] \oplus (\mathtt{S}, i) = G_l^{(i)} \oplus (\mathtt{S}, i) = G_{l+1}^{(i)} \oplus (\mathtt{S}, i) = \ldots = G_{k-1}^{(i)} \oplus (\mathtt{S}, i) = G_k^{(i)} \enspace .$$

Consider next the case $v_k \in V_{ij}^R$ and let $v_m = \omega(v_k)$; then $\chi(v_k) = (\mathtt{R}, i, j)$ and $\chi(v_m) = (\mathtt{S}, j)$. As we follow the scheduling scheme of Definition 26, the pairs (ad, c') of v_m and v_k coincide. Correspondingly the abort condition in line 19 of Recv_c does not trigger, the pairs (ad', c) of v_m and v_k match, and so do the strings I_j of v_m (before being overwritten in line 12 of Send_c) and $\iota^1 \,\|\, \ldots \,\|\, \iota^t$ of v_k. Lemma 1(b) and the correctness of Ch now imply that also the messages m considered in v_m and v_k are equal, and that the abort condition in line 24 of Recv_c is not met. We show $h[v_k] \neq \perp$ below (i.e., the abort condition in line 31 of Recv_c is not triggered), and hence this establishes $m[v_k] = \mu(\omega(v_k))$.

It remains to show that $h[v_k] = G_k^{(i)}$. Concerning participant j, observe that the strings I_j compiled in line 12 of Send_c and line 27 of Recv_c are slices of the projected characteristic $\chi_j(G)$ (see Definition 18 on page 51) and correspond precisely with those considered in Lemma 8(a) (on page 51). Now, if v_k is the smallest (i.e., first) action in V_i then the incoming string $\iota^1 \,\|\, \ldots \,\|\, \iota^t = I_j$ is empty (i.e., $t = 0$), by (CAUS); in particular we have $h[v_k] = (\emptyset, \emptyset, \emptyset) \oplus (\mathtt{S}, j) \oplus (\mathtt{R}, i, j) = G_k^{(i)}$. Let otherwise v_l be the (unique) direct predecessor of v_k in V_i. Then $l < k$ and for all $l < r < k$ we have $v_r \notin V_i$. By the inductive assumption we have $h[v_l] = G_l^{(i)}$. Further, if we write $\mathcal{S}$ as shortcut for $(\mathtt{R}, j, \iota^1) \oplus \ldots \oplus (\mathtt{R}, j, \iota^t) \oplus (\mathtt{S}, j) \oplus (\mathtt{R}, i, j)$ and apply Lemma 8(b) sufficiently often, we finally obtain

$$h[v_k] = h[v_l] \oplus \mathcal{S} = G_l^{(i)} \oplus \mathcal{S} = G_{l+1}^{(i)} \oplus \mathcal{S} = \ldots = G_{k-1}^{(i)} \oplus \mathcal{S} = G_k^{(i)} \enspace .$$

□

We proceed with proving that message and ciphertext integrity are lifted from Ch to Ch_c, and that message integrity of the former suffices to obtain history integrity of the latter. These results are stated in Theorem 11. We then show that confidentiality against passive adversaries is lifted from Ch to Ch_c, as stated in Theorem 12. The results above together with Theorem 5

(on page 60) culminate in Corollary 2, which identifies sufficient security requirements on the FIFO channel Ch for obtaining a causal channel Ch_c that offers the strongest security properties defined in this thesis.

Theorem 11 (Integrity of Construction 4)**.** *If* Ch *offers (FIFO) integrity of ciphertexts, respectively, of messages, then* Ch_c *offers (causal) integrity of ciphertexts, resp., of messages. More precisely, for all efficient adversaries* $\mathcal{A}$ *and* $\mathcal{A}'$ *against* Ch_c *there exist efficient adversaries* $\mathcal{B}$ *and* $\mathcal{B}'$ *against* Ch *such that*

$$\mathbf{Adv}^{\text{C-INT-PTXT}}_{\mathsf{Ch}_c,N,\mathcal{A}}(\lambda) \leq \mathbf{Adv}^{\text{F-INT-PTXT}}_{\mathsf{Ch},N,\mathcal{B}}(\lambda) \quad \textit{and} \quad \mathbf{Adv}^{\text{C-INT-CTXT}}_{\mathsf{Ch}_c,N,\mathcal{A}'}(\lambda) \leq \mathbf{Adv}^{\text{F-INT-CTXT}}_{\mathsf{Ch},N,\mathcal{B}'}(\lambda) \ .$$

In addition, if Ch *offers message integrity then* Ch_c *provides history integrity. Precisely, for every efficient adversary* $\mathcal{A}''$ *against* Ch_c *there exists an efficient* $\mathcal{B}''$ *such that*

$$\mathbf{Adv}^{\text{C-INT-HIST}}_{\mathsf{Ch}_c,N,\mathcal{A}''}(\lambda) \leq \mathbf{Adv}^{\text{F-INT-PTXT}}_{\mathsf{Ch},N,\mathcal{B}''}(\lambda) \ .$$

Proof. We prove the three statements left-to-right. A copy of the C-INT-PTXT experiment from Figure 6.4, with the algorithms of our construction from Figure 6.5 plugged in, is given as E^0 in Figure 6.6 (ignore the commented out lines 21–22, 32–33, and 43-44 for now). We added some variable assignments in lines 02, 07, 10, 13–14, 19, 23, 25, 31, 40, and 45 that do not affect the outcome of the experiment but help in the analysis below.

Let $\mathrm{E}^1_{\mathcal{A}}$ be the modification of $\mathrm{E}^0_{\mathcal{A}}$ with lines 32–33 activated, as indicated in the figure. Observe that the setup of the Send operation of line 12 and the Recv operation of line 29 (these are of the FIFO channel Ch) together with the accompanying code in lines 02, 13–14 and 30–33 precisely corresponds with the arrangement of the F-INT-PTXT experiment from Figure 5.3. This means that line 33 will only be executed if adversary $\mathcal{A}$ (implicitly) breaks the message integrity of Ch. Thus a standard reductionist argument shows that $\Pr[\mathrm{E}^0_{\mathcal{A}} = 1] \leq \mathbf{Adv}^{\text{F-INT-PTXT}}_{\mathsf{Ch},N,\mathcal{B}}$, for an efficient adversary $\mathcal{B}$.

In an execution of experiment E^1 with adversary $\mathcal{A}$, consider the sequence $G_0, G_1, \ldots$ of intermediate snapshots of graph G as they occur throughout the experiment (lines 07, 19, 40). Observe that after any execution of lines 19 and 40 the then current graph G_n is a prefix of the then current graph $\bar{G}$ from lines 13 and 31 (prefix in the sense of that sequence $\chi_i(G_n)$ is a prefix of sequence $\chi_i(\bar{G})$, for all $i \in [1\,..\,N]$). Indeed, every $\oplus$ operation in line 18 is preceded by a corresponding $+$ operation in line 13 and every $\oplus$ operation in line 39 is a preceded by a corresponding $+$ operation in line 31, and if in any oracle invocation the $+$ operation succeeds but the corresponding $\oplus$ operation fails then no further action of the affected participant is ever executed. By consequence, the nodes v and $\bar{v}$ of lines 18 and 13 naturally correspond, and so do the nodes from lines 39 and 31; concerning the latter, in particular also nodes $\omega(\bar{v})$ and $\omega(v)$ correspond.

Let us now define a function ψ such that $\psi\colon ((ad, I), m) \mapsto (ad, m)$. Then concerning line 41 we have $(ad, m) = \psi(ad', m) = \psi(\bar{Q}[\omega(\bar{v})]) = Q[\omega(v)]$, where we used lines 28, 32, 14, 11, and then 20. That is, to complete the proof of the C-INT-PTXT property it remains to show that in line 41 always $G_n \neq \bot$. We prove this condition below, as part of the C-INT-HIST proof.

Concerning the second statement, which is on the C-INT-CTXT security of Ch_c, observe that the new F-INT-CTXT assumption on Ch is taken into account by changing lines 14 and 32 of experiments $\mathrm{E}^0_{\mathcal{A}}, \mathrm{E}^1_{\mathcal{A}}$ to consider (ad', c') instead of (ad', m). The remaining steps are identical to the ones given above, but with function ψ replaced by (almost) the identity function. Indeed, for the (updated) line 41 we would then have $(ad, c) = (ad, (I, c')) = ((ad, I), c') = (ad', c') = (\bar{Q}[\omega(\bar{v})]) = Q[\omega(v)]$.

We move on to the third statement, i.e., the claim that Ch_c offers C-INT-HIST security if Ch is F-INT-PTXT-secure. Consider the corresponding experiment E^2 as defined in Figure 6.6

$\mathrm{E}^0_{\mathcal{A}}, \mathrm{E}^1_{\mathcal{A}}, \mathrm{E}^2_{\mathcal{A}}(1^\lambda)$
01 $G \leftarrow (\emptyset,\emptyset,\emptyset)$, $Q[\,] \leftarrow \emptyset$
02 $\bar{G} \leftarrow (\emptyset,\emptyset,\emptyset)$, $\bar{Q}[\,] \leftarrow \emptyset$
03 $(\hat{st}_1,\ldots,\hat{st}_N) \leftarrow_{\$} \mathsf{Init}(1^\lambda, N)$
04 For $i \leftarrow 1$ to N:
05 $\quad I_i \leftarrow \varepsilon$
06 $\quad h_i \leftarrow (\emptyset,\emptyset,\emptyset)$
07 $G_0 \leftarrow G$, $H_0 \leftarrow G$, $n \leftarrow 1$
08 $\mathcal{A}^{\mathcal{O}_{\mathsf{Send}},\mathcal{O}_{\mathsf{Recv}}}(1^\lambda)$
09 Terminate with 0

If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Send}}(i, ad, m)$:
10 $x_n \leftarrow (\mathtt{S}, i)$
11 $ad' \leftarrow (ad, I_i)$
12 $(\hat{st}_i, c') \leftarrow_{\$} \mathsf{Send}(\hat{st}_i, ad', m)$
13 $\bar{G} \leftarrow \bar{G} + (\mathtt{S}, i)$ with $\bar{v}$
14 $Q[\bar{v}] \leftarrow (ad', m)$
15 $c \leftarrow (I_i, c')$
16 $I_i \leftarrow \epsilon$
17 $h_i \leftarrow h_i \oplus (\mathtt{S}, i)$
18 $G \leftarrow G \oplus (\mathtt{S}, i)$ with v
19 $G_n \leftarrow G$, $H_n \leftarrow h_i$
20 $Q[v] \leftarrow (ad, m)$
21 // If $G = \bot \vee h_i \neq G^{(i)}$: ($\mathrm{E}^2$)
22 // $\quad$ Terminate with 1 (E^2)
23 $n \leftarrow n + 1$
24 Return (c, h_i) to $\mathcal{A}$

If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Recv}}(i, j, ad, c)$:
25 $x_n \leftarrow (\mathtt{R}, i, j)$
26 Parse c as $(\iota^1 \| \ldots \| \iota^t, c')$
27 If parsing fails: Return $\bot$ to $\mathcal{A}$
28 $ad' \leftarrow (ad, \iota^1 \| \ldots \| \iota^t)$
29 $(\hat{st}_i, m) \leftarrow_{\$} \mathsf{Recv}(\hat{st}_i, j, ad', c')$
30 If $m = \bot$: Return $\bot$ to $\mathcal{A}$
31 $\bar{G} \leftarrow \bar{G} + (\mathtt{R}, i, j)$ with $\bar{v}$
32 // If $\bar{G} = \bot \vee (ad', m) \neq \bar{Q}[\omega(\bar{v})]$: ($\mathrm{E}^1, \mathrm{E}^2$)
33 // $\quad$ Terminate with 0 ($\mathrm{E}^1, \mathrm{E}^2$)
34 $I_i \leftarrow I_i \| j$
35 For $k \leftarrow 1$ to t:
36 $\quad h_i \leftarrow h_i \oplus (\mathtt{R}, j, \iota^k)$
37 $h_i \leftarrow h_i \oplus (\mathtt{S}, j) \oplus (\mathtt{R}, i, j)$
38 If $h_i = \bot$: Return $\bot$ to $\mathcal{A}$
39 $G \leftarrow G \oplus (\mathtt{R}, i, j)$ with v
40 $G_n \leftarrow G$, $H_n \leftarrow h_i$
41 If $G = \bot \vee (ad, m) \neq Q[\omega(v)]$: ($\mathrm{E}^0, \mathrm{E}^1$)
42 $\quad$ Terminate with 1 ($\mathrm{E}^0, \mathrm{E}^1$)
43 // If $G = \bot \vee h_i \neq G^{(i)}$: ($\mathrm{E}^2$)
44 // $\quad$ Terminate with 1 (E^2)
45 $n \leftarrow n + 1$
46 Return (m, h_i) to $\mathcal{A}$

Figure 6.6: *Experiments* $\mathrm{E}^0, \mathrm{E}^1, \mathrm{E}^2$ *used in the proofs for C-INT-CTXT, C-INT-PTXT, and C-INT-HIST of Construction 4.*

(here, lines 21–22, 32–33, and 43–44 are activated, but lines 41–42 are deactivated. As above, the difference between E^2 and the original C-INT-HIST game from Figure 6.4 is bounded by $\mathbf{Adv}^{\text{F-INT-PTXT}}_{\mathsf{Ch},N,\mathcal{B}''}(\lambda)$, for an efficient adversary $\mathcal{B}''$.

Consider an execution of adversary $\mathcal{A}$ in E^2 and the corresponding sequence of graphs G_n and H_n throughout the game. We show by induction on n that the conditions of lines 21 and 43 are never met. For some n let thus $(i, j) \in [\![1\,..\,N]\!]$ be such that $x_n = (\mathtt{S}, i)$ or $x_n = (\mathtt{R}, i, j)$ (lines 10 and 25) and $G_m \neq \bot$ and $G_{n-1} \neq \bot$ and $H_m = G^{(i)}_m$, where m is minimal such that $0 \leq m < n$ and $x_{m+1}, \ldots, x_{n-1} \notin X_i$, i.e., either $m = 0$ or x_m is the action of participant i that directly precedes x_n. We need to prove $G_n \neq \bot$ and $H_n = G^{(i)}_n$ (in lines 21 and 43).

Consider first the case $x_n = (\mathtt{S}, i)$. We have $G_n = G_{n-1} \oplus x_n \neq \bot$ by line 18 and Definition 17. Further, from line 17 and sufficiently many applications of Lemma 8(b) we derive

$$H_n = H_m \oplus (\mathtt{S}, i) = G^{(i)}_m \oplus (\mathtt{S}, i) = G^{(i)}_{m+1} \oplus (\mathtt{S}, i) = \ldots = G^{(i)}_{n-1} \oplus (\mathtt{S}, i) = G^{(i)}_n \ .$$

Consider next the case $x_n = (\mathtt{R}, i, j)$. Write $G_{n-1} = (V, \leq, \chi)$. To establish $G_n \neq \bot$, by Definition 17 it suffices to show that there exists $u \in V^S_j \setminus {}^{\leq}V_i$ such that ${}^{<}u \cap V^S \subseteq {}^{\leq}V_i$. Let $s = |V^S_j|$ and $r = |V^R_{ij}|$. By the observations above we know that G_{n-1} is a prefix of $\bar{G}$ and thus $s \geq r + 1$ by Lemma 1(b), where the '+1' is due to the fact that action x_n is not considered in G_{n-1}. Let $u \in V^S_j$ be the $(r+1)$-th sending action in V_j. Then $u \notin {}^{\leq}V_i$ by Lemma 1(c). We

next show that for all $w \in {}^{<}u \cap V^S$ we have $w \in {}^{\leq}V_i$. If $w \in V_i^S$ the statement is clear. Further, if $w \in V_j^S$ then w is at most the r-th sending action in V_j and hence $w \in {}^{\leq}V_i$ by Lemma 1(c). It remains to analyze the case $w \in V_l^S$ with $l \in [1\,..\,N] \setminus \{i, j\}$.

Let $I = \iota^1 \parallel \ldots \parallel \iota^t$ be the string from line 26 of the $\mathcal{O}^*_{\mathsf{Recv}}$ invocation corresponding to x_n and let $v_1 \prec_\ell \ldots \prec_\ell v_{t'} \prec_\ell u$ be the longest subchain of V_j such that $v_k \in V^R$ for all $1 \leq k \leq t'$. As I is transported in the associated data field of Ch (line 28), and as u and x_n are the $(r+1)$-th actions with characteristics $(\mathtt{S}, j)$ and $(\mathtt{R}, i, j)$, respectively, that appear in $\bar{G}$, by lines 32–33, 16 and 34 we have $t = t'$ and $\chi(v_k) = (\mathtt{R}, j, \iota^k)$ for all k.

Now, towards proving that for $w \in V_l{}^S$ we have $w \in {}^{\leq}V_i$, note that by $w < u$ there exists a minimal $w^* \in V_j^R$ such that $w < w^* \prec_\ell u$; by (CAUS) we even have $w \prec_r w^* \prec_\ell u$. If $w^* \prec_\ell v_1$ then by the definition of v_1 there exists $\hat{w} \in V_j^S$ with $w^* \prec_\ell \hat{w} \prec_\ell v_1$. As $\hat{w}$ is at most the r-th sending action in V_j we have $\hat{w} \in {}^{\leq}V_i$ by Lemma 1(c), and hence also $w \in {}^{\leq}V_i$.

Only one case is remaining: $w^* = v_k$ with $1 \leq k \leq t'$, i.e., $\iota^k = l$. Write $H_m = (W, \leq, \chi)$. As $H_n = H_m \oplus (\mathtt{R}, j, \iota^1) \oplus \ldots \oplus (\mathtt{R}, j, \iota^t) \oplus (\mathtt{S}, j) \oplus (\mathtt{R}, i, j) \neq \bot$ by lines 35–38, also for $\bar{H}_m = H_m \oplus (\mathtt{R}, j, \iota^1) \oplus \ldots \oplus (\mathtt{R}, j, \iota^k)$ we have $\bar{H}_m \neq \bot$. Write $\bar{H}_m = (\bar{W}, \leq, \chi)$ and let $r' = |\bar{W}_{jl}^R|$. By Lemma 1(b) we have that w is the r'-th action in $\bar{W}_l^S = W_l^S$. We apply Lemma 3(b) to $H_m = G_m^{(i)}$ and deduce $|W_{il}^R| = |W_l^S| \geq r'$. That is, by Lemma 1(b) there exists $w' \in W_{il}^R$ such that $w \prec_r w'$. Thus $w \in {}^{\leq}W_{il}^R \subseteq {}^{\leq}V_i$. This completes the proof of $G_n = G_{n-1} \oplus x_n \neq \bot$. Observe that $H_n = G_n^{(i)}$ now follows immediately from Lemma 8(b).

All in all, the conditions from lines 21 and 43 never hold and lines 22 and 44 will not be executed. This shows that Ch_c offers C-INT-HIST security. □

Theorem 12 (Confidentiality against passive adversaries)**.** *If* Ch *offers indistinguishability under chosen-plaintext attacks (F-IND-CPA) then* Ch_c *offers causal indistinguishability under chosen-plaintext attacks (C-IND-CPA). More precisely, for every efficient adversary* $\mathcal{A}$ *against* Ch_c *there exists an efficient adversary* $\mathcal{B}$ *against* Ch *such that*

$$\mathbf{Adv}^{\text{C-IND-CPA}}_{\mathsf{Ch}_c, N, \mathcal{A}}(\lambda) \leq \mathbf{Adv}^{\text{F-IND-CPA}}_{\mathsf{Ch}, N, \mathcal{B}}(\lambda) \ .$$

Proof. Let $\mathcal{A}$ be an adversary that plays the C-IND-CPA game against channel Ch_c and let $\mathcal{B}$ be the reduction specified in Figure 6.7. To facilitate the comparison between the C-IND-CPA experiment that $\mathcal{B}$ simulates for $\mathcal{A}$ and the F-IND-CPA experiment that $\mathcal{B}$ plays against, in Figure 6.7 we reproduce, along with $\mathcal{B}$'s code, the instructions executed within $\mathcal{O}_{\mathsf{LoR}}$ and $\mathcal{O}^*_{\mathsf{Recv}}$. We marked these additional instructions with the comment symbol '// ' to make a clear distinction between what happens in the 'outer' F-IND-CPA game and what $\mathcal{B}$ actually does. Notice that for each of $\mathcal{A}$'s left-or-right and receiving queries, $\mathcal{B}$ poses a corresponding query to the oracles $\mathcal{O}_{\mathsf{LoR}}$ and $\mathcal{O}^*_{\mathsf{Recv}}$ provided by the F-IND-CPA experiment. Precisely, if $\mathcal{A}$ asks to send (i, ad, m^0, m^1) then $\mathcal{B}$ queries $\mathcal{O}_{\mathsf{LoR}}$ with (i, ad', m^0, m^1) where $ad' = (ad, I_i)$; let c' be the oracle answer; then $\mathcal{B}$ returns the pair (h, c), where $c' = (I_i, c)$, to $\mathcal{A}$. Similarly, if $\mathcal{A}$ poses a receiving query (i, j, ad, c) then $\mathcal{B}$ queries $\mathcal{O}^*_{\mathsf{Recv}}$ on tuple (i, j, ad', c') where $c = (\iota^1 \parallel \ldots \parallel \iota^t, c')$ and $ad' = (ad, \iota^1 \parallel \ldots \parallel \iota^t)$. Let φ be the bijection that maps $(ad, (I, c')) \mapsto ((ad, I), c')$. Using φ we can make the relation between $\mathcal{A}$'s and $\mathcal{B}$'s receiving queries explicit: any query (i, j, ad, c) posed by $\mathcal{A}$ corresponds to a query $(i, j, \varphi(ad, c))$ posed by $\mathcal{B}$.

Let G, respectively, $\bar{G}$ be causal graph and the FIFO graph recorded by $\mathcal{B}$ while emulating the C-IND-CPA game and considered within the execution of the F-IND-CPA experiment that $\mathcal{B}$ plays, respectively. During the experiment execution, G and $\bar{G}$ are expanded according to the communication scheduled by $\mathcal{A}$: each left-or-right query causes the addition (via $\oplus$, respectively, $+$) of a sending node to G and to $\bar{G}$, respectively, while each (passive) receiving query adds a receiving node to G and to $\bar{G}$, respectively. Observe that, although $\bar{G}$ is expanded using the FIFO addition $+$ while for G we use the causal addition $\oplus$, the two graphs evolve

$\mathcal{B}^{\mathcal{O}_{\mathsf{LoR}},\mathcal{O}^*_{\mathsf{Recv}}}(1^\lambda)$:
01 $G \leftarrow (\emptyset,\emptyset,\emptyset), Q[\,] \leftarrow \emptyset$
02 // $\bar{G} \leftarrow (\emptyset,\emptyset,\emptyset), \bar{Q}[\,] \leftarrow \emptyset$
03 For $i \leftarrow 1$ to N:
04 $\quad I_i \leftarrow \epsilon$
05 $\quad h_i \leftarrow (\emptyset,\emptyset,\emptyset)$
06 $b' \leftarrow_{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{LoR}},\mathcal{O}^*_{\mathsf{Recv}}}(1^\lambda)$
07 Terminate with b'

If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{LoR}}(i, ad, m^0, m^1)$:
08 Require $|m^0| = |m^1|$
09 $ad' \leftarrow (ad, I_i)$
10 $c' \leftarrow \mathcal{O}_{\mathsf{LoR}}(i, ad', m^0, m^1)$
11 // Require $|m^0| = |m^1|$
12 // $(\hat{st}_i, c') \leftarrow \mathsf{Send}(\hat{st}_i, ad', m^b)$
13 // $\bar{G} \leftarrow \bar{G} + (\mathtt{S}, i)$ with $\bar{v}$
14 // $\bar{Q}[\bar{v}] \leftarrow (ad', c')$
15 $c \leftarrow (I_i, c')$
16 $I_i \leftarrow \epsilon$
17 $h_i \leftarrow h_i \oplus (\mathtt{S}, i)$
18 $G \leftarrow G \oplus (\mathtt{S}, i)$ with v
19 $Q[v] \leftarrow (ad, c)$
20 Return (c, h_i) to $\mathcal{A}$

If $\mathcal{A}$ queries $\mathcal{O}^*_{\mathsf{Recv}}(i, j, ad, c)$:
21 $G \leftarrow G \oplus (\mathtt{R}, i, j)$ with v
22 If $G = \perp \vee (ad, c) \neq Q[\omega(v)]$:
23 $\quad$ Terminate with 0
24 Parse c as $(\iota^1 \,\|\, \ldots \,\|\, \iota^t, c')$
25 If parsing fails: Terminate with 0
26 $ad' \leftarrow (ad, \iota^1 \,\|\, \ldots \,\|\, \iota^t)$
27 $\diamond \leftarrow \mathcal{O}^*_{\mathsf{Recv}}(i, j, ad', c')$
28 // $\bar{G} \leftarrow \bar{G} + (\mathtt{R}, i, j)$ with $\bar{v}$
29 // If $\bar{G} = \perp \vee (ad', c') \neq \bar{Q}[\omega(\bar{v})]$:
30 // $\quad$ Terminate with 0
31 // $(\hat{st}_i, m) \leftarrow \mathsf{Recv}(\hat{st}_i, j, ad', c')$
32 $I_i \leftarrow I_i \,\|\, j$
33 For $k \leftarrow 1$ to t:
34 $\quad h_i \leftarrow h_i \oplus (\mathtt{R}, j, \iota^k)$
35 $h_i \leftarrow h_i \oplus (\mathtt{S}, j) \oplus (\mathtt{R}, i, j)$
36 If $h_i = \perp$: Terminate with 0
37 Return $(\diamond, h_i)$ to $\mathcal{A}$

Figure 6.7: *Security reduction $\mathcal{B}$ used in the proof for C-IND-CPA of Construction 4.*

simultaneously, action by action. Hence, at any point in time when a query is processed and the next query is not yet asked, the graphs G and $\bar{G}$ are isomorphic. More precisely, let $G = (V, \leq, \chi)$ and $\bar{G} = (\bar{V}, \bar{\leq}, \bar{\chi})$ denote the graphs obtained after processing any of $\mathcal{A}$'s query. Then, there exists a bijection

$$\phi\colon \bar{V} \to V;\ \bar{v} \mapsto v \quad \text{such that} \quad \chi(\phi(\bar{v})) = \chi(\bar{v}) \quad \text{and} \quad \phi(\omega(\bar{v})) = \omega(\phi(\bar{v}))\ .$$

In order to bound $\mathcal{A}$'s advantage with $\mathcal{B}$'s advantage it remains to show that, if $\mathcal{A}$ does not cause the C-IND-CPA game to terminate abruptly, neither does $\mathcal{B}$ in the F-IND-CPA game. In this regard, things can go wrong only when the receiving oracle is invoked. Thus, we only analyze the latter case. Assume that all of $\mathcal{A}$'s queries are passive (in a causal sense), i.e., they do not trigger the execution of instructions 23, 25 and 30. We show next that the corresponding queries posed by $\mathcal{B}$ are passive (in a FIFO sense) as well. Let G and $\bar{G}$ be the 'current' (isomorphic) causal and FIFO graphs and suppose that $\mathcal{A}$ asks a query $q = (i, j, ad, c)$ to $\mathcal{O}^*_{\mathsf{Recv}}$. By the assumption of passiveness of $\mathcal{A}$ we know that q makes the operation $G \oplus (\mathtt{R}, i, j)$ admissible and, if v denotes the node newly added to G, we have $(ad, c) = Q[\omega(v)]$. Our assumption also guarantees that the parsing operation of line 24 does not fail. Write $c = (\iota^1 \,\|\, \ldots \,\|\, \iota^t, c')$ and $ad' = (ad, \iota^1 \,\|\, \ldots \,\|\, \iota^t)$. We now prove that query $q' = (i, j, ad', c')$, which $\mathcal{B}$ asks to $\mathcal{O}^*_{\mathsf{Recv}}$, is passive in a FIFO sense. As $\bar{G}$ and G are isomorphic, if follows from lemma 6 (Chapter 4) that $G \oplus (\mathtt{R}, i, j) \neq \perp \Longrightarrow \bar{G} + (\mathtt{R}, i, j) \neq \perp$. Let $\bar{v}$ be the node added to $\bar{G}$. We have

$$(ad', c') \overset{(1)}{=} \varphi(ad, c) \overset{(2)}{=} \varphi(Q[\omega(v)]) \overset{(3)}{=} \varphi(Q[\phi(\omega(\bar{v}))]) \overset{(4)}{=} \bar{Q}[\omega(\bar{v})]\ ,$$

where the first equality holds by definition of φ, the second is implied by the previously established relation $(ad, c) = Q[\omega(v)]$, the third follows from the relations $v = \phi(\bar{v})$ and $\omega(\phi(\bar{v})) = \phi(\omega(\bar{v}))$, and the fourth is implied by the following observation: for all sending actions $\bar{w} \in \bar{V}$ and $w \in V$ such that $w = \phi(\bar{w})$, the C-IND-CPA experiment stores pair $Q[w] = (x, y)$ if and only if the F-IND-CPA experiment stores pair $\bar{Q}[\bar{w}] = \varphi(x, y)$. This proves that instruction 30 is not executed. Hence, if the C-IND-CPA game does not penalize $\mathcal{A}$ then the F-IND-CPA does not penalize $\mathcal{B}$ either. □

Corollary 2 (Security of Construction 6.5)**.** *If* Ch *offers integrity of ciphertexts and indistinguishability under chosen-plaintext attacks (i.e., F-INT-CTXT and F-IND-CPA) then* Ch_c *offers the strongest security guarantees for causal channels: causal ciphertext integrity, history integrity, and causal indistinguishability under chosen-ciphertext attacks (i.e., C-INT-CTXT, C-INT-HIST, and C-IND-CCA).*

Chapter 7

Sequential Key Generators

In the present and the next chapter we leave the domain of channels and focus on schemes that deterministically expand a random seed into a sequence of random-looking keys. Such schemes, that we name sequential key generators, have been shown to be a powerful tool for building forward-secure symmetric primitives. In this chapter we propose an improved security model for sequential key generators and discuss how it compares with existing models.

7.1 Introduction

A cryptosystem provides *forward security* (FS) if it continues to give meaningful security guarantees after the adversary got a copy of the used keys. A standard example is key exchange: here, all recent security models require established session keys to remain safe when the adversary obtains access to the involved long-term private keys [Sho99, CK01]. The notion of forward security also extends to non-interactive primitives. For instance, in forward-secure public key encryption [CHK07] messages are encrypted in respect to a combination (pk, t), where pk is a public key and $t \in \mathbb{N}$ identifies one out of a set of consecutive time epochs; for each such epoch t, knowledge of a specific decryption key sk_t is necessary for decrypting corresponding ciphertexts. In addition, while by design it is efficiently possible to perform updates $sk_t \mapsto sk_{t+1}$, forward security requires that the reverse mapping be inefficient, i.e., it shall be infeasible to 'go backwards in time'. More precisely, forward security guarantees that plaintexts encrypted for 'expired' epochs remain confidential even if the decryption keys of all later epochs are revealed. Analogously to the described setting, signatures produced using the forward-secure variants of signature schemes remain unforgeable for past epochs if only current and future keys are disclosed to the adversary [BM99].

In the symmetric-key setting, forward security for encryption and authentication primitives is defined similarly (clearly, here there is no public key). For instance, Bellare and Yee [BY03] show that one way to obtain a forward-secure encryption scheme is to combine a (forward-secure) *sequential key generator* (SKG) with a regular encryption scheme, where the former can be seen as a PRG that maintains state and, once initialized with a random seed, deterministically outputs a pseudorandom sequence of random-looking keys. These keys are then used, one per epoch, together with the encryption scheme to ensure indistinguishability of ciphertexts within each time epochs. It seems plausible that forward-secure channels can be boot-strapped using the same modular approach, by combining a 'regular' channel protocol with a forward-secure sequential key generator. Note that for symmetric-key primitives two or more parties (e.g., the users of a FIFO channel, Chapter 5) need to maintain a copy of the same secret key. To achieve forward security, the communicating parties may run in synchrony the same instance

of a sequential key generator and use matching keys for encryption and decryption. From the perspective of an attacker, this scenario opens the possibility to obtain encryption keys of the generated sequence from either of the participants, potentially in an adaptive way if the key updates are not performed at the same time.[1]

In the present chapter we propose a new security model for SKGs that reflects the scenario described above where an attacker may have adaptive access to the key sequence. Sequential key generators also find applications in computer forensic, particularly in the context of secured local logging.

7.2 Syntax and Functionality

Informally speaking, a *sequential key generator* (SKG) is a stateful primitive that deterministically expands an initial seed, chosen uniformly at random, into a sequence of fix-length keys. These keys are supposed to be used in higher level protocols, for example as keys for symmetric encryption or message authentication schemes. More specifically, an SKG consists of a parameter generation algorithm Gen that selects public parameters suitable for any security level specified by the user, an initialization algorithm Init that takes as input the public parameters and generates an initial state, a key derivation algorithm GetKey that, given as input the current epoch's state, derives the corresponding key, and a state update algorithm Evolve that also takes as input the current epoch's state and outputs the state for the next epoch.

Definition 29 (Syntax of SKGs)**.** *Let $\ell\colon \mathbb{N} \to \mathbb{N}$ be a polynomial. A* sequential key generator *with key length ℓ is a tuple of efficient algorithms $\mathsf{SKG} = (\mathsf{Gen}, \mathsf{Init}, \mathsf{Evolve}, \mathsf{GetKey})$ as follows:*

- Gen*. This randomized algorithm takes as input a security parameter 1^λ and the number of supported epochs $T \in \mathbb{N} \cup \{\infty\}$, and outputs public parameters par and optionally some auxiliary information aux. We write this as $(par, aux) \leftarrow_\$ \mathsf{Gen}(1^\lambda, T)$.*
- Init*. This randomized algorithm takes as input the public parameters par and returns an initial state st_0. We denote this by $st_0 \leftarrow_\$ \mathsf{Init}(par)$.*
- Evolve*. This algorithm takes as input the current state st_i and outputs the next state st_{i+1}. To indicate this we write $st_{i+1} \leftarrow \mathsf{Evolve}(st_i)$ and use the shortcut Evolve^k to denote the k-fold composition of Evolve, i.e., $st_{i+k} \leftarrow \mathsf{Evolve}^k(st_i)$, for every $k \in \mathbb{N}$.*
- GetKey*. This algorithm extracts from any state st_i the key $K_i \in \{0,1\}^{\ell(\lambda)}$. We write this as $K_i \leftarrow \mathsf{GetKey}(st_i)$. We use the shortcut $\mathsf{GetKey}^k(st_i)$ to denote $\mathsf{GetKey}(\mathsf{Evolve}^k(st_i))$.*

Note that our syntax allows algorithm Gen to receive as input $T = \infty$. In fact, some schemes do not require the maximum number of epochs to be specified in advance as they may support, in principle, an unlimited number of iterations (this is indeed the case for the generic construction that we present in Section 8.4.1). For these schemes we omit the input value T from the signature of Gen (i.e., when the number of epochs is not specified we assume $T = \infty$). In fact, we anticipate that all schemes presented in this thesis offer security only if T is not 'too large' (precisely: it must be polynomial in the security parameter).

In some practical scenarios, SKG instances are not run as a single copy. When used to secure local logging services, for instance, after clones of an initial state st_0 are distributed to a given set of parties, several copies of the same SKG instance may be run concurrently and independently, potentially on different host systems not necessarily in synchronization. As Evolve and GetKey

[1] Another application that inherently requires at least two parties to use the same instance of an SKG is local logging, as we will see later.

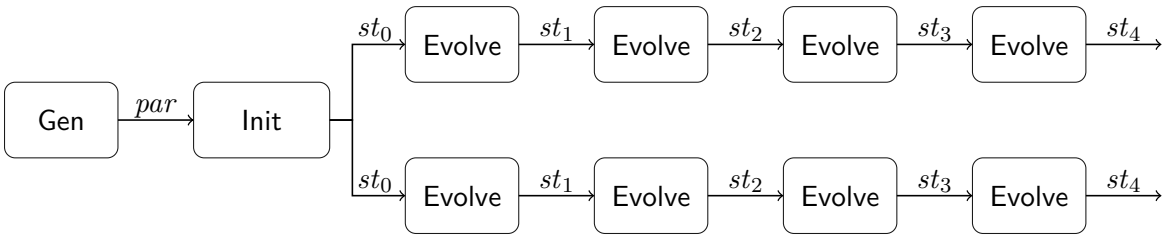


Figure 7.1: *Interplay of the SKG algorithms* Gen, Init, *and* Evolve. *The figure shows two copies of the same SKG instance running in parallel.* GetKey *algorithm can be applied to each intermediate state* st_i *to derive key* K_i.

are deterministic, respective state and key sequences of the same SKG instance are identical for all copies. This setting is illustrated in Figure 7.1.

7.3 Security

Similarly to PRGs, we require indistinguishability of generated keys from random strings of the same length as basic security property of SKGs. Our security model considers an experiment involving an adversary $\mathcal{A}$ who first gets adaptive access to a set of real keys K_i of her choosing (with 'real' keys we mean keys generated by running an instance of the SKG algorithms) and is then challenged with a string K_n^b that is either the real key K_n or a random string of the same length; the adversary has to distinguish these two cases. This shall model the intuition that key K_n 'looks random' even if the adversary is given all other keys K_i, for $i \neq n$. We formalize indistinguishability in the following definition.

Definition 30 (IND security for SKGs)**.** *A sequential key generator* SKG *provides* indistinguishability against adaptive adversaries *(IND) if for all efficient adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *that interact in the experiment* $\mathrm{Expt}^{\mathrm{IND},b}_{\mathsf{SKG},\mathcal{A}}$ *from Figure 7.2 the following advantage function*

$$\mathbf{Adv}^{\mathrm{IND}}_{\mathsf{SKG},\mathcal{A}}(\lambda) := \left|\Pr\left[\mathrm{Expt}^{\mathrm{IND},1}_{\mathsf{SKG},\mathcal{A}}(1^\lambda) = 1\right] - \Pr\left[\mathrm{Expt}^{\mathrm{IND},0}_{\mathsf{SKG},\mathcal{A}}(1^\lambda) = 1\right]\right| \ ,$$

is negligible, where the probabilities are taken over the random coins of the experiment, including $\mathcal{A}$*'s randomness.*

Several applications of SKGs have to cope with the possibility that an adversary eventually gets the state of the system (e.g., by means of computer break-in) and hence is able to compute all subsequent keys. While nothing can be done in such settings to protect the keys generated after corruption took place, we can still hope that previously generated keys remain secure. For these applications we demand a stronger security guarantee than indistinguishability that also incorporates *forward security*. We model break-in by additionally letting the adversary corrupt any state of the generator. More precisely, starting from the IND security experiment we incorporate forward security by allowing $\mathcal{A}$ to specify together with the challenge epoch n also a corruption epoch m and obtain the generator state for that epoch st_m. Clearly, knowledge of st_m gives the ability to compute all subsequent keys $K_m, K_{m+1}, \ldots$, thus, to exclude trivial wins we must require that the challenge epoch precedes the corruption epoch. We formalize indistinguishability with forward security in the following definition.

$\mathrm{Expt}^{\mathrm{IND},b}_{\mathsf{SKG},T,\mathcal{A}}(1^\lambda)$:
01 $(par, aux) \leftarrow_\$ \mathsf{Gen}(1^\lambda, T)$
02 $st_0 \leftarrow_\$ \mathsf{Init}(par)$
03 $S_{keyid} \leftarrow \emptyset$
04 $(hist, n) \leftarrow_\$ \mathcal{A}_1^{\mathcal{O}_{\mathsf{GetKey}}(\cdot)}(1^\lambda, T)$
05 Require $0 \le n < T$
06 $K_n^0 \leftarrow_\$ \{0,1\}^{\ell(\lambda)}$
07 $K_n^1 \leftarrow \mathsf{GetKey}^n(st_0)$
08 $b' \leftarrow_\$ \mathcal{A}_2^{\mathcal{O}_{\mathsf{GetKey}}(\cdot)}(hist, K_n^b)$
09 Require $n \notin S_{keyid}$
10 Return b'

If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{GetKey}}(i)$:
11 Require $0 \le i < T$
12 $S_{keyid} \leftarrow S_{keyid} \cup \{i\}$
13 $K_i \leftarrow \mathsf{GetKey}^i(st_0)$
14 Return K_i to $\mathcal{A}$

Figure 7.2: *Security experiments of* indistinguishability *(IND) for sequential key generators. The auxiliary output aux does not play any role in the security game and can be ignored for now. Set S_{keyid} is used to register the epochs for which the adversary requests the SKG key. The adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ operates in two stages and maintains a state variable hist that allows $\mathcal{A}_1$ to pass information to $\mathcal{A}_2$.*

Definition 31 (IND-FS security for SKGs)**.** *A sequential key generator* SKG *is* indistinguishable with forward security against adaptive adversaries *(IND-FS) if for all efficient adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ that interact in the experiment $\mathrm{Expt}^{\text{IND-FS},b}_{\mathsf{SKG},\mathcal{A}}$ from Figure 7.3 the following advantage function*

$$\mathbf{Adv}^{\text{IND-FS}}_{\mathsf{SKG},\mathcal{A}}(\lambda) := \left|\Pr\left[\mathrm{Expt}^{\text{IND-FS},1}_{\mathsf{SKG},\mathcal{A}}(1^\lambda) = 1\right] - \Pr\left[\mathrm{Expt}^{\text{IND-FS},0}_{\mathsf{SKG},\mathcal{A}}(1^\lambda) = 1\right]\right| \ ,$$

is negligible, where the probabilities are taken over the random choices of the experiment, including $\mathcal{A}$'s randomness.

We pose the informal requirement on Evolve algorithm that it securely erase state st_i after deriving state st_{i+1} from it. Note that secure erasure is generally considered difficult to achieve and requires special care [Gut96].

It is immediate to see that the IND-FS notion is strictly stronger than the IND notion.

7.4 Comparison with Stateful Generators

Stateful generators, first described by Bellare and Yee (BY) [BY03, Section 2.2], aim at similar applications as SKGs. The two primitives are essentially identical (where 'essentially' indicates a purely syntactical difference in the algorithm specifications). This is true for what is concerned with the functionality of SKGs. Regarding security, the BY model and ours present significant differences that we highlight next.

In the security experiment for stateful generators (a.k.a. *forward-secure pseudorandom generators*, or FS-PRG for short), after having incremental access to a sequence $K_0, K_1, \ldots$ of keys that are either all real (i.e., $K_i \leftarrow \mathsf{GetKey}(st_i)\ \forall i$) or all random (i.e., $K_i \leftarrow_\$ \{0,1\}^{\ell(\lambda)}\ \forall i$), the adversary eventually requests to see the 'current' state st_m and, based upon the result, outputs a guess on whether keys $K_0, \ldots, K_{m-1}$ were actually real or random. For reference, we reproduce in Figure 7.4 the model from [BY03], adjusted to our notation and syntax.

An important difference between the BY model and ours is that for FS-PRGs an adversary that corrupts state st_m *cannot* request access to keys K_i, $i \ge m$, before corruption takes place

$\mathrm{Expt}^{\text{IND-FS},b}_{\mathsf{SKG},T,\mathcal{A}}(1^\lambda)$:
01 $(par, aux) \leftarrow_\$ \mathsf{Gen}(1^\lambda, T)$
02 $st_0 \leftarrow_\$ \mathsf{Init}(par)$
03 $S_{keyid} \leftarrow \emptyset$
04 $(hist, n, m) \leftarrow_\$ \mathcal{A}_1^{\mathcal{O}_{\mathsf{GetKey}}(\cdot)}(1^\lambda, T)$
05 Require $0 \le n < m < T$
06 $K_n^0 \leftarrow_\$ \{0,1\}^{\ell(\lambda)}$
07 $K_n^1 \leftarrow \mathsf{GetKey}^n(st_0)$
08 $st_m \leftarrow \mathsf{Evolve}^m(st_0)$
09 $b' \leftarrow_\$ \mathcal{A}_2^{\mathcal{O}_{\mathsf{GetKey}}(\cdot)}(hist, st_m, K_n^b)$
10 Require $n \notin S_{keyid}$
11 Terminate with b'

If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{GetKey}}(i)$:
12 Require $0 \le i < T$
13 $S_{keyid} \leftarrow S_{keyid} \cup \{i\}$
14 $K_i \leftarrow \mathsf{GetKey}^i(st_0)$
15 Return K_i to $\mathcal{A}$

Figure 7.3: *Security experiments of* indistinguishability with forward security *(IND-FS) for sequential key generators. The auxiliary output aux does not play any role in the security game and can be ignored for now. Set S_{keyid} is used to register the epochs for which the adversary requests the SKG key. The adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ operates in two stages and maintains a state variable hist for allowing $\mathcal{A}_1$ to pass information to $\mathcal{A}_2$.*

(in contrast to our model). This difference may be limiting in contexts where multiple parties evolve states of the same SKG instance independently of each other and in an asynchronous manner. For instance, in the secure logging scenario, the adversary might *first* observe the log auditor verifying MAC tags on 'current' time epochs and *then* decide to corrupt a monitored host that is out of synchronization, e.g., because it is powered down and hence didn't evolve its state. As such concurrent and asynchronous conditions are not considered in the model by Bellare and Yee, in some practically relevant settings the security of the constructions from [BY03] should not be assumed.

We note, however, that denying the adversary access to keys K_i, $i \ge m$ in the BY model is necessary to avoid trivial attacks: by seeing K_m and then obtaining st_m, an attacker may simply compare K_m with $\mathsf{GetKey}(st_m)$ and, in case the two keys coincide, infer that the observed keys were real, otherwise that they were random. It seems that our model, instead, excludes such trivial attack by letting the adversary obtain all real keys except for the one challenge key K_n^b. Here, seeing first key K_m and *then* corrupting state st_m is perfectly fine. This observation raises a natural question: is IND-FS security strictly stronger than FS-PRG security?

As we prove in the following, it turns out that, surprisingly, the two security notions are equivalent. We prove first that our model is at least as strong as BY's model, i.e., IND-FS $\Longrightarrow$ FS-PRG. Intuitively, the BY security experiment can be seen as a restriction of ours in which the adversary in the first phase of the experiment requests to see keys $K_1, \ldots, K_{n-1}$ and then declares challenge epoch n and corruption epoch $m = n + 1$. Recall that in the BY experiment the challenger gives the adversary either the real keys $K_1, \ldots, K_n$ or randomly chosen strings of the same length, depending on the value of the hidden bit b. In contrast, the challenger of the IND-FS experiment always provides the adversary with real keys $K_1, \ldots, K_{n-1}$ and only lets the challenge key K_n^b to be real, if $b = 1$, or random, otherwise. Shortly: in the BY experiment keys $K_1, \ldots, K_{n-1}, K_n$ are distributed as $\mathrm{Re}/\$, \ldots, \mathrm{Re}/\$, \mathrm{Re}/\$$ while in our model we have $\mathrm{Re}, \ldots, \mathrm{Re}, \mathrm{Re}/\$$.

Theorem 13 (IND-FS $\Longrightarrow$ FS-PRG)**.** *Let* SKG *be a stateful key generator supporting T iter-*

$\mathrm{Expt}^{\text{FS-PRG},b}_{\mathsf{SKG},\mathcal{A}}(1^\lambda)$:
01 $(par, aux) \leftarrow_\$ \mathsf{Gen}(1^\lambda, T)$
02 $i \leftarrow 0, hist \leftarrow \varepsilon$
03 $st_0 \leftarrow_\$ \mathsf{Init}(par)$
04 Do:
05 $\quad K_i^0 \leftarrow_\$ \{0,1\}^{\ell(\lambda)}$
06 $\quad K_i^1 \leftarrow_\$ \mathsf{GetKey}(st_i)$
07 $\quad (d, hist) \leftarrow_\$ \mathcal{A}(\mathsf{find}, K_i^b, hist)$
08 $\quad st_{i+1} \leftarrow \mathsf{Evolve}(st_i)$
09 $\quad i \leftarrow i + 1$
10 While $d = \mathsf{find}$ and $i < T - 1$
11 $b' \leftarrow_\$ \mathcal{A}(\mathsf{guess}, st_i, hist)$
12 Terminate with b'

Figure 7.4: *Security experiment of forward security for stateful generators (FS-PRG) introduced by Bellare and Yee [BY03], adapted to our syntax. We denote by hist a history variable, initially empty, maintained by the adversary between different invocations. Flags* find *and* guess *indicates two 'modes' of the adversary, referring to the phase in which $\mathcal{A}$ observes keys (before corruption) and the phase in which $\mathcal{A}$ expresses its verdict on the nature of the seen keys (after corruption). Note that we count time-epochs from 0 to $T - 1$.*

ations. If SKG *offers IND-FS security then it also provides FS-PRG security. Moreover, for every efficient adversary $\mathcal{A}$ against the FS-PRG property there exists an efficient adversary $\mathcal{B}$ against the IND-FS property such that*

$$\mathbf{Adv}^{\text{FS-PRG}}_{\mathsf{SKG},\mathcal{A}}(\lambda) \leq (T-1) \cdot \mathbf{Adv}^{\text{IND-FS}}_{\mathsf{SKG},\mathcal{B}}(\lambda) \ .$$

Proof. Let $\mathrm{E}^{0,b}_{\mathcal{A}}$ denote the FS-PRG security experiment involving adversary $\mathcal{A}$ against SKG with the hidden bit is set to b: Our goal is to bound the following advantage:

$$\mathbf{Adv}^{\text{FS-PRG}}_{\mathsf{SKG},\mathcal{A}}(\lambda) = \left|\Pr\left[\mathrm{E}^{0,1}_{\mathcal{A}}(1^\lambda) = 1\right] - \Pr\left[\mathrm{E}^{0,0}_{\mathcal{A}}(1^\lambda) = 1\right]\right| \ ,$$

with the IND-FS-advantage of some efficient algorithm $\mathcal{B}$.

Let n denote the number of keys that the adversary sees before corruption takes place (i.e., $n = i$ when the loop of lines 04–10 terminates). We proceed by defining a sequence of hybrids $\mathrm{H}^{j,b}_{\mathcal{A}}$ for $j \in [0\,..\,n]$ where $\mathrm{H}^{0,b}_{\mathcal{A}} := \mathrm{E}^{1,b}_{\mathcal{A}}$ and $\mathrm{H}^{j+1,b}_{\mathcal{A}}$ is derived from $\mathrm{H}^{j,b}_{\mathcal{A}}$ by turning K_j^0 into a 'real' key, i.e., $K_j^0 \leftarrow \mathsf{GetKey}(st_j)$ (in particular, this implies that $K_j^0 = K_j^1$). Throughout the proof let $\Pr[\mathrm{E}^{j,b}_{\mathcal{A}}]$ and $\Pr[\mathrm{H}^{j,b}_{\mathcal{A}}]$ be shortcuts for the probabilities $\Pr[\mathrm{E}^{j,b}_{\mathcal{A}}(1^\lambda) = 1]$ and $\Pr[\mathrm{H}^{j,b}_{\mathcal{A}}(1^\lambda) = 1]$, for $j \in \mathbb{N}$ and $b \in \{0,1\}$.

It follows directly by construction that $\Pr[\mathrm{H}^{j,1}_{\mathcal{A}}] = \Pr[\mathrm{E}^{0,1}_{\mathcal{A}}]$ for all $j \in [0\,..\,n]$ (for $b = 1$ the adversary sees only the 'left' keys, which are not affected by the changes between hybrids). Regarding the $b = 0$ case, we claim that for every $j \in [0\,..\,n-1]$ there exists an efficient IND-FS-adversary $\mathcal{B}_j$ such that:

$$\left|\Pr\left[\mathrm{H}^{j,0}_{\mathcal{A}}\right] - \Pr\left[\mathrm{H}^{j+1,0}_{\mathcal{A}}\right]\right| \leq \mathbf{Adv}^{\text{IND-FS}}_{\mathsf{SKG},\mathcal{B}_j}(\lambda) \ .$$

Algorithm $\mathcal{B}_j$ emulates the game for $\mathcal{A}$ by computing the keys and the corrupt state as follows. It gets keys $K_1, \ldots, K_{j-1}$ from oracle $\mathcal{O}_{\mathsf{GetKey}}$, requests the j-th key as challenge key K_j^b, chooses

the remaining keys $K_{j+1}, \ldots, K_n$ uniformly at random from $\{0,1\}^{\ell(\lambda)}$, and requests as corrupt state $st_m = st_{n+1}$. Eventually $\mathcal{B}_j$ returns the same output as $\mathcal{A}$.

Next, we show that also for the last hybrid $\mathrm{H}_{\mathcal{A}}^{n,b}$ there exists an efficient IND-FS adversary $\mathcal{B}_n$ such that:

$$\left|\Pr\left[\mathrm{H}_{\mathcal{A}}^{n,1}\right] - \Pr\left[\mathrm{H}_{\mathcal{A}}^{n,0}\right]\right| \leq \mathbf{Adv}^{\text{IND-FS}}_{\mathsf{SKG},\mathcal{B}_n}(\lambda) \ .$$

Similarly to previous algorithms $\mathcal{B}_j$, $\mathcal{B}_n$ obtains keys $K_1, \ldots, K_{n-1}$ from the oracle $\mathcal{O}_{\mathsf{GetKey}}$, requests challenge epoch n and corruption epoch $n+1$ and gets back challenge key K_n^b and state st_{n+1}, hence provides $\mathcal{A}$ with keys $K_1, \ldots, K_{n-1}, K_n^b$ and state st_{n+1}. Finally, it outputs the same bit b' that $\mathcal{A}$ returns.

Now let $\mathcal{B}$ be the algorithm that first chooses j uniformly at random from $[0\,..\,T-2]$ (recall that we can assume $n < m$ and $m \in [0\,..\,T-1]$ wlog in the IND-FS experiment) and then emulates the FS-PRG game for $\mathcal{A}$ by running $\mathcal{B}_j$ with the only exception that if $i \neq j$ when $\mathcal{A}$ outputs $d = \mathsf{guess}$ (recall that $i = n$ when the loop of lines 04–10 terminates) then $\mathcal{B}$ terminates the simulation with output 0; otherwise, it proceeds as $\mathcal{B}_j$ does. Since j is randomly chosen, we have that $\mathbf{Adv}^{\text{IND-FS}}_{\mathsf{SKG},\mathcal{B}}(\lambda) = \frac{1}{T-1} \cdot \sum_{j=0}^{T-2} \mathbf{Adv}^{\text{IND-FS}}_{\mathsf{SKG},\mathcal{B}_j}(\lambda)$.

Putting all together we obtain the final bound:

$$\mathbf{Adv}^{\text{FS-PRG}}_{\mathsf{SKG},\mathcal{A}}(\lambda) \leq (T-1) \cdot \mathbf{Adv}^{\text{IND-FS}}_{\mathsf{SKG},\mathcal{B}}(\lambda) \ .$$

□

The opposite implication, FS-PRG $\Longrightarrow$ IND-FS, can be proven using a similar strategy as in the proof of Theorem 13. We give only a proof idea, the details are easy to infer from the previous proof. If n denotes the epoch that an IND-FS adversary $\mathcal{A}$ requests as 'challenge', we first proceed via a hybrid argument that progressively turns all keys $K_1, \ldots, K_{n-1}$ into 'challenge keys', i.e., as for key K_n they are computed using the SKG algorithms if $b = 1$, while they are chosen uniformly at random in $\{0,1\}^{\ell(\lambda)}$ in case $b = 0$. Shortly, this step changes the distributions of the keys (up to the challenge key) from $\mathrm{Re}, \ldots, \mathrm{Re}, \mathrm{Re}/\$$ to $\mathrm{Re}/\$, \ldots, \mathrm{Re}/\$, \mathrm{Re}/\$$. It is then easy to simulate the resulting game using an FS-PRG adversary that guesses the challenge epoch, requests (in advance) keys $K_1, \ldots, K_n$ and corrupt state st_{n+1} from the FS-PRG experiment, hence answers $\mathcal{A}$'s queries $i, i < n$ to $\mathcal{O}_{\mathsf{GetKey}}$ by returning the obtained keys if $i < n$ and by computing $K_i = \mathsf{GetKey}(st_i)$ for $i > n$, returns K_n as challenge key, and computes the corrupt state as $st_m = \mathsf{Evolve}(st_n)^{m-n}$.

This leads us to the following result which, together with the statement of Theorem 13, establishes the equivalence between the IND-FS and the FS-PRG security notions.

Theorem 14 (FS-PRG $\Longrightarrow$ IND-FS)**.** *Let* SKG *be a stateful key generator supporting* T *iterations. If* SKG *offers FS-PRG security then it also provides IND-FS security. Moreover, for every efficient adversary* $\mathcal{A}$ *against the FS-PRG property there exists an efficient adversary* $\mathcal{B}$ *against the IND-FS property such that*

$$\mathbf{Adv}^{\text{IND-FS}}_{\mathsf{SKG},\mathcal{A}}(\lambda) \leq (T-1) \cdot \mathbf{Adv}^{\text{FS-PRG}}_{\mathsf{SKG},\mathcal{A}}(\lambda) \ .$$

7.5 Constructions

Sequential key generators can be easily built from standard cryptographic primitives. Bellare and Yee (BY) propose some constructions from PRGs and PRFs, and prove their forward security (FS-PRG) in [BY03]. For concreteness, we reproduce a simple design of an SKG based on PRGs; its FS-PRG security is analyzed in [BY03, Theorem 1]. The IND-FS security of $\mathsf{PRG\text{-}SKG}$ is a direct corollary of Theorem 14 (on page 93). Briefly, the generator expands an

initial seed via a PRG invocation and then splits the output into two parts, the next epoch's seed (this is indeed the running state of the generator) and the current epoch's key.

Construction 5 (SKGs from PRGs, [BY03]). *Let $\ell\colon \mathbb{N} \to \mathbb{N}$ be a positive polynomial and let G be a PRG such that $|G(x)| = |x| + \ell(|x|)$ for all x. Write $G(x)$ as $G_L(x) \| G_R(x)$ with $|G_L(x)| = \lambda$ and $|G_R(x)| = \ell(\lambda)$ and define* $\mathsf{PRG\text{-}SKG} = (\mathsf{Gen}, \mathsf{Init}, \mathsf{Evolve}, \mathsf{GetKey})$ *by letting* Gen *output the empty string,* Init *sample $st_0 \leftarrow_{\$} \{0,1\}^\lambda$,* $\mathsf{Evolve}(st_i)$ *output $G_L(st_i)$, and* $\mathsf{GetKey}(st_i)$ *output $G_R(st_i)$.*

Theorem 15 ([BY03]). *Let $T\colon \mathbb{N} \to \mathbb{N}$ be a polynomial and let G be a PRG as in Construction 5. The stateful generator* PRG-SKG *supporting $T(\lambda)$ epochs offers FS-PRG security.*

We will see more constructions of sequential key generators in the next chapter.

7.6 A Digression on Secured Local Logging

Computer log files can be configured to record a large variety of system events that occur on network hosts and communication systems, including users logging on or off, memory resources reaching their capacity, malfunctioning of disk drives, etc. Therefore, log files represent an essential source of information that support system administrators in understanding the activity of systems and keeping them fully functional. Unfortunately, as log files are often recorded locally (i.e., on the monitored machine itself), in many practical cases intruders can a posteriori manipulate the log entries related to their attacks. In a network environment, one obvious strategy to prevent adversarial tampering of audit logs is to forward log messages immediately after their creation to a remote log sink—in the hope that the attacker cannot also corrupt the latter. Necessary in such a setting is that the log sink is continuously available, as every otherwise required local buffering of log records would increase the risk that their delivery is suppressed by the adversary. However, in many cases the reachability of the log sink can be artificially restrained by the intruder, e.g., by confusing routing protocols with false ARP messages, by sabotaging TCP connections with injected reset packets, by jamming wireless connections, or by directing application-level denial-of-service attacks against the log sink.

A solution for tamper-resistant log-entry storage that does not require a remote log sink but offers integrity protection via cryptographic means is *secured local logging*. Here, each log entry is stored together with a specific authentication tag that is generated and verified using a secret key. Note that regular message authentication codes (MACs) by themselves seem not to constitute a secure solution, as corresponding tags will be forgeable by intruders that succeed in extracting the secret key from the attacked device. Rather, the forward-secure variant of a MAC is required. The latter can be realized by combining a regular MAC with a sequential key generator. An early approach towards secured local logging originates from Bellare and Yee [BY97]; they study the role of forward security in authentication, develop the security notion of *forward integrity*, and realize a corresponding primitive via a PRF chain. Shortly after [BY97], an independent cryptographic scheme specifically targeted at protecting log files was described by Kelsey and Schneier [KS98, KS99, SK99]. Their scheme draws its (forward) security from frequent key updates via iterated hashing, but is unfortunately not supported by a formal security analysis. Kelsey, Callas, and Clemm [KCC10] introduced secured logging into the standardization process at IETF. However, their proposal of *signed syslog messages* focuses on remote logging instead of on local logging. Precisely, their extension to the standard UNIX *syslog* facility authenticates log entries via signatures before sending them to a log sink over the network. While this proposal naturally offers seekability, it is bound to the full-time availability of an online log sink. Indeed, periods where the latter is not reachable are not securely covered, as the scheme is not forward-secure.

Chapter 8

Seekable Sequential Key Generators

In this chapter we augment the notion of a sequential key generator by adding a fast-forward functionality to compute any key of the output sequence from the generator's initial state. This property finds a natural application in the context of secured local logging.

8.1 Introduction

We have seen in the previous chapter that forward-secure SKGs are not difficult to construct. For instance, the scheme of Bellare and Yee from Construction 5 is an efficient one. Indeed, if it is instantiated with a hash function-based PRG, invocations of Evolve and GetKey algorithms take only a small (constant) number of hash function evaluations. However, this assessment of efficiency is adequate only if the derived keys K_i are used (and computed) in sequential order. We argue that in many potential fields of application such access structures are not given; instead, random access to the keys may be required, implying a considerable efficiency penalty if keys need to be computed iteratively via $K_i \leftarrow \mathsf{GetKey}^i(st_0)$. The following examples illustrate that random access patterns do not intrinsically contradict the envisioned sequential nature of SKGs.

Consider a host that uses SKG's keys K_i to authenticate continuously incurring log messages. A second copy of the same SKG instance would be run by the log auditor. From time to time the latter might want to check the integrity of an arbitrary selection of these messages. Observe that this scenario does not really correspond to the setting from Figure 7.1: While the upper SKG copy might represent the host that evolves keys in the expected linear order $K_i \rightarrow K_{i+1}$, the auditor (running the second copy) would actually need non-sequential access to SKG's keys.

For a second example in secure logging, assume SKG's epochs are coupled to absolute time intervals (e.g., one epoch per second). If a host is powered up after a long down-time, in order to resynchronize its SKG state, it is required to do a 'fast-forward' over a large number of epochs. Ideally, an SKG would support the option to skip an arbitrary number of Evolve steps. Clearly, a (fast-)forward algorithm with execution time linear in the number k of skipped epochs is trivially achievable. The question is: can we do better than $O(k)$? We next introduce a variant of SKG that explicitly offers random access capabilities.

8.2 Seekability

Recall that a sequential key generator (as well as a stateful PRG) can be seen as a stateful primitive that outputs a sequence of random-looking keys—one per invocation. The distinguished property of *seekability* ensures that it is possible to jump directly to any position in

the output sequence. Observe that knowledge of the initial state of a (regular) SKG instance does permit to derive every key of the output sequence. However, deriving the ith key from state st_0 requires to iterate i times the update procedure Evolve, passing through all intermediate states $st_1, \ldots, st_i$ and then invoking GetKey on state st_i to derive key K_i. Instead, a *seekable* SKG (shortly: SSKG) provides a fast-forward procedure Seek that allows a designated party who owns a so-called seeking key to derive from the initial state any subsequent state without, and more efficiently than, repeatedly invoking the update procedure Evolve through all the intermediate states.

Definition 32 (Seekable SKGs)**.** *Let* $\mathsf{SKG} = (\mathsf{Gen}, \mathsf{Init}, \mathsf{Evolve}, \mathsf{GetKey})$ *be a sequential key generator. We say that* SKG *is* seekable *if the parameter generation algorithm* Gen *outputs as auxiliary information a so-called* seeking key *sk and there exists an efficient deterministic algorithm* Seek *that given as inputs the seeking key sk, an initial state* st_0*, and an integer* $k \in \mathbb{N}$*, returns an updated state* st_k*, that we write as* $st_k \leftarrow \mathsf{Seek}(sk, st_0, k)$*, with the following property: for all* $\lambda \in \mathbb{N}$*, all* $T \in \mathbb{N}$*, all* $(par, sk) \leftarrow_{\$} \mathsf{Gen}(1^\lambda, T)$*, all* $st_0 \leftarrow_{\$} \mathsf{Init}(par)$*, and all integers* $0 \leq k \leq \infty$ *it holds* $\mathsf{Seek}(sk, st_0, k) = \mathsf{Evolve}^k(st_0)$*. In this case we say that the tuple* $\mathsf{SSKG} = (\mathsf{Gen}, \mathsf{Init}, \mathsf{Evolve}, \mathsf{GetKey}, \mathsf{Seek})$ *is a* seekable sequential key generator *(SSKG).*

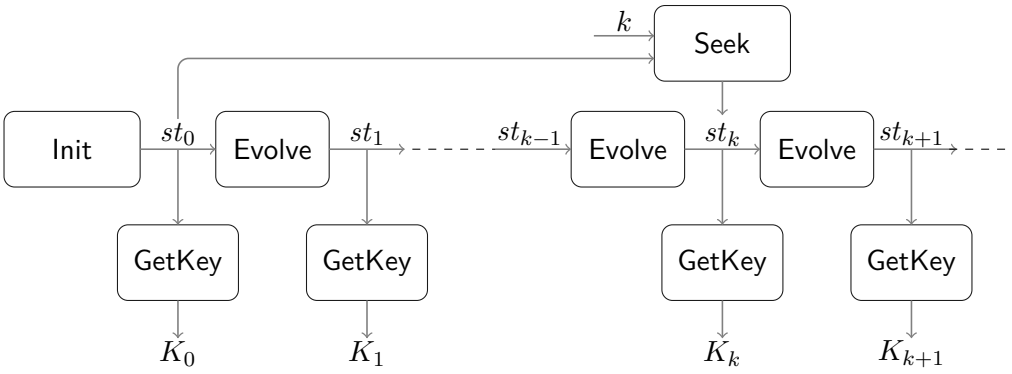


Figure 8.1: *Interplay of the different SSKG algorithms. Given the seeking key sk and the initial state* st_0*, one can seek directly to any arbitrary state* st_k*.*

Many practical applications can widely benefit from the extended functionality. Observe that the advantage of seekable SKGs over (regular) SKGs is purely efficiency-wise; in particular, the definition of SSKG security is almost identical to the one for SKGs (with 'almost' we mean 'modulo syntactical changes').

Remark 6 (Epochs outside of supported range)*.* Note that the correctness requirement leaves unspecified the effect of Evolve when invoked from st_0 for more than T times and of Seek for an unsupported epoch number $k \geq T$. Clearly, if the application demands it, an SSKG can always be implemented such that Evolve, Seek, and GetKey output an error symbol for such epochs, simply by including an epoch counter in the state. We abstain from formally requiring such a behavior as our construction in Section 8.4.2 in many cases does guarantee security for a larger number of epochs than requested and there seems to be no reason to generally disregard systems that offer this extra service.

Remark 7 (On the necessity of seeking trapdoors)*.* Observe that for standard SKGs the secret material managed by users in each time-epoch i is restricted to the one current state st_i. In contrast, for SSKGs we introduced additional secret information, namely the seeking key sk, required to perform the Seek operation. One might ask whether this step was really necessary. We fixed the syntax of SSKGs as given in Definition 32 for a technical reason: the SSKG

constructions that we present in Section 8.4.1 are factoring-based, respectively, RSA-based, and the corresponding Seek algorithms require knowledge of the modulus factorization $n = pq$ (the shortcut is $\varphi = (p-1)(q-1)$). However, as knowledge of p and q thwarts the one-wayness of designated Evolve operation, we had to formally separate the entities that can and cannot perform the Seek operation. While this property slightly narrows the applicability of SSKGs, it is irrelevant for the intended secure logging scenario that we aim at, described below.

Application of SSKGs: secured local logging. We describe a practical setting of secured local logging with multiple monitored hosts. The system administrator first runs Gen algorithm to establish system-wide parameters; each host then runs Init algorithm to create its individual initial state st_0, serving as a basis for specific sequences $(st_i)_{i\in\mathbb{N}}$ and $(K_i)_{i\in\mathbb{N}}$. The log auditor, having access to seeking key sk and to initial states st_0 of all hosts, can reproduce all corresponding keys K_i without restriction. Observe that, as the SSKG instances on different hosts are independent of each other (i.e., each host creates its own initial state and, thus, derives a key chain that is independent of chains created by other hosts), authenticated log messages from one host cannot be 'replayed' on other hosts.

In practice, it might be difficult to find 'the right' frequency with which keys should be evolved to the next epoch. Recall that, even if forward-secure log authentication is in place, an intruder cannot be prevented from manipulating the log entries of the epoch in which he got access to a system. This suggests that keys should be updated at least every few seconds—and even more often to obtain protection against fully-automated attack tools.

8.3 Shortcut One-Way Permutations

We introduce a novel primitive, called *shortcut (one-way) permutation* (ScP), that will serve as a building block for our generic SSKG construction presented in Section 8.4.1. Consider a finite set D together with an efficiently computable permutation $\pi\colon D \to D$. Clearly, for any $x \in D$ and $m \in \mathbb{N}$ one can compute the m-fold composition $\pi^m(x) = \pi \circ \cdots \circ \pi(x)$ by evaluating m-times the permutation π; the latter can be done in linear time $\mathcal{O}(m)$. We define shortcut permutations by requiring the efficiency feature that the value $\pi^m(x)$ can be computed more efficiently than that, using a dedicated algorithm. In addition we demand one-wayness of π: given the image $y = \pi(x) \in D$ of a uniformly chosen element $x \in D$ it should be computationally hard to compute x.

While we will rigorously specify the one-wayness requirement of ScPs, we do not give a precise definition of what 'more efficiently' means for the computation of π^m. The reason is that we aim at practicality of our construction, and, in general, practical efficiency strongly depends on the concrete parameter sizes and computing platforms in use.

We formalize next the syntax and functionality of ScPs. For technical reasons, the definition slightly deviates from the above intuition in that the algorithm which efficiently computes π^m also requires an auxiliary input, that we call the *shortcut* information; moreover, it defines shortcut one-way permutations as families of functions (Chapter 2 on page 9).

Definition 33 (Shortcut Permutations)**.** *A* shortcut permutation *is a tuple* $\mathsf{ScP} = (\mathsf{ScPGen}, \mathsf{Eval}, \mathsf{Express})$ *of efficient algorithms as follows:*

- *The* parameter generation *algorithm* ScPGen *takes as input a security parameter* 1^λ *and outputs a set of public parameters* P *together with a shortcut information* sc*. We write this as* $(P, sc) \leftarrow_\$ \mathsf{ScPGen}(1^\lambda)$*. We assume that each specific value* P *implicitly defines a finite domain* D_P*, and that elements from* D_P *can be efficiently sampled with uniform distribution.*

- *The* evaluation *algorithm* Eval *takes as input the public parameters* P*, an element* $x \in D_P$*, and returns an element* $y \in D_P$*. We write this as* $y \leftarrow \mathsf{Eval}(P, x)$*.*

- *The* shortcut *algorithm* $\mathsf{Express}$ *takes as input the shortcut information* sc*, an element* $x \in D_P$*, an integer* $m \in \mathbb{N}$*, and returns an element* $y \in D_P$*. We write* $y \leftarrow \mathsf{Express}(sc, x, m)$*.*

We require that for all $\lambda \in \mathbb{N}$ *and all* $(P, sc) \leftarrow_{\$} \mathsf{ScPGen}(1^\lambda)$ *the algorithm* $\mathsf{Eval}(P, \cdot)$ *implements a bijection* $\pi\colon D_P \to D_P$ *and that* $\mathsf{Express}(sc, x, m) = \pi^m(x)$ *for all* $x \in D_P$ *and* $m \in \mathbb{N}$*. Additionally we demand that for all* $x \in D_P$ *and all* $m > 1$ *computing* $\mathsf{Express}(sc, x, m)$ *be 'significantly faster' than* $\mathsf{Eval}(P, x)^m$*.*

As the shortcut property is solely an efficiency feature, it does not appear in our specification of one-way security. In fact, the one-wayness requirement of ScPs and of regular one-way permutations (as defined in Chapter 2) are essentially the same. Importantly, we do not grant the adversary access to the shortcut.

Definition 34 (One-wayness of ScP)**.** *Let* $\mathsf{ScP} = (\mathsf{ScPGen}, \mathsf{Eval}, \mathsf{Express})$ *be a shortcut permutation. We say that* ScP *is* one way *if for all efficient adversaries* $\mathcal{A}$ *the following advantage function is negligible:*

$$\mathbf{Adv}^{\mathrm{OWP}}_{\mathsf{ScP},\mathcal{A}}(\lambda) := \Pr\left[\mathsf{Eval}(P, x) = y \;:\; (P, sc) \leftarrow_{\$} \mathsf{ScPGen}(1^\lambda);\; y \leftarrow_{\$} D_P;\; x \leftarrow_{\$} \mathcal{A}(P, y)\right]$$

where the probabilities are taken over the random choices of the experiment and $\mathcal{A}$*'s randomness.*

Remark 8 (Shortcut *vs* Trapdoor One-Way Permutations)*.* The syntax of ScPs is, to some extent, close to that of trapdoor permutations (TDPs, see [KL15]). However, observe the significant difference between the notions of *trapdoor* and *shortcut*. While a TDP's trapdoor allows efficient *inversion* of the permutation (i.e., computation of π^{-1}), a shortcut allows efficient computation of the composition π^m, for arbitrary m. In particular for some ScPs it may be the case that, even when the shortcut information is available, there is no way to efficiently invert π. We admit, though, that in our constructions described from the next section, one-wayness does not hold for adversaries that obtain the shortcut information: in fact, any party knowing the shortcut can also efficiently invert the permutation.

We propose two efficient constructions of ScPs, $\mathsf{FACT\text{-}ScP}$ and $\mathsf{RSA\text{-}ScP}$, that are one-way based on the RSA problem and on the SQRT problem (Definitions 1–2 on page 8).

Construction 6 ($\mathsf{FACT\text{-}ScP}$)**.** *Let* $\mathsf{SQRTGen}$ *be an SQRT generation algorithm and define the shortcut permutation* $\mathsf{FACT\text{-}ScP} = (\mathsf{ScPGen}, \mathsf{Eval}, \mathsf{Express})$ *as follows. Let* $\mathsf{ScPGen}(1^\lambda)$ *run* $(N, p, q, \varphi) \leftarrow_{\$} \mathsf{SQRTGen}(1^\lambda)$ *and output* $P = N$ *and* $sc = \varphi$*. For every* $x \in \mathbb{QR}_N$ *let* $\mathsf{Eval}(N, x)$ *output* $x^2 \bmod N$*, and for every* $m \in \mathbb{N}$ *let* $\mathsf{Express}(\varphi, x, m)$ *output* $x^{(2^m \bmod \varphi)} \bmod N$*.*

Remark 9 ($\mathsf{FACT\text{-}ScP}$ is a shortcut one-way permutation)*.* For every security parameter λ and every tuple (N, p, q, φ) generated by $\mathsf{SQRTGen}(1^\lambda)$ let $D_P := \mathbb{QR}_N$ and let $\pi\colon D_P \to D_P$ be the mapping $x \mapsto x^2 \bmod N$. Observe that the specified domain D_P is efficiently samplable: it suffices to take $x \leftarrow_{\$} \mathbb{Z}_N^*$ uniformly at random and then square it. By definition, $\mathsf{Eval}(P, \cdot)$ implements the bijection $\pi\colon D_P \to D_P$. It follows from standard number-theoretic results (in particular [MvV97, Fact 2.160] and [MvV97, Fact 2.126]) that for every $x \in D_P$ and every $m \in \mathbb{N}$ it holds $\mathsf{Express}(\varphi, x, m) = \mathsf{Eval}^m(N, x)$. Every $\mathsf{Express}$ operation takes about one exponentiation modulo N. Further, comparing the experiments in Definition 34 (on page 98) and Definition 1 (on page 8) makes it evident that $\mathsf{FACT\text{-}ScP}$ is one way if the SQRT problem is hard for $\mathsf{SQRTGen}$, i.e., if integer factorization is hard [MvV97, Fact 3.46].

Construction 7 (RSA-ScP)**.** *Let* RSAGen *be an RSA generation algorithm and define the shortcut permutation* RSA-ScP = (ScPGen, Eval, Express) *as follows. Let* $\mathsf{ScPGen}(1^\lambda)$ *run* $(N, \varphi, e) \leftarrow_\$ \mathsf{RSAGen}(1^\lambda)$ *and output* $P = (N, e)$ *and* $sc = \varphi$*. For every* $x \in \mathbb{Z}_N$ *let* $\mathsf{Eval}(N, e, x)$ *output* $x^e \bmod N$*, and for every* $m \in \mathbb{N}$ *let* $\mathsf{Express}(\varphi, x, m)$ *output* $x^{(e^m \bmod \varphi)} \bmod N$*.*

Remark 10 (RSA-ScP is a shortcut one-way permutation). For any $(N, \varphi, e) \leftarrow_\$ \mathsf{RSAGen}(1^\lambda)$ let $D_P := \mathbb{Z}_N$ and let $\pi\colon D_P \to D_P$ be the mapping $x \mapsto x^e \bmod N$. Using a similar argument as in Remark 9 we conclude that the domain D_P is efficiently samplable, that correctness of the ScP follows from standard number-theoretic results, and that every Express operation takes about one exponentiation modulo N. Further, RSA-ScP is one way as long as the RSA problem is hard for RSAGen.

Factoring- ***vs*** **RSA-based shortcut permutations.** Observe that both constructions rely on different, though related, number-theoretic assumptions. In fact, while the security of FACT-ScP can be shown to be equivalent to the hardness of integer factorization, RSA-ScP can be reduced 'only' to the RSA assumption. Hence, in some sense, SQRT-based schemes are at least as secure as RSA-based schemes. In addition to that, our SQRT-based scheme has a (slight) performance advantage over our RSA-based scheme (squaring is more efficient than raising to the power of e). The only situation we are aware of in which RSA-ScP might have an advantage over FACT-ScP is when the most often executed operation is Express, and deployment of multiprime RSA is acceptable (e.g., $N = pqr$). Briefly, in the multiprime RSA setting [JK03, HLT03], private key operations can be implemented particularly efficiently, based on the Chinese Remainder Theorem.

8.4 Constructions

In this section we construct sequential key generators that offer a seekability feature. Our first construction is based on shortcut one-way permutations and enjoys a proof of security in the random oracle model. Since shortcut one-way permutation can be based on the hardness of factoring and of RSA, this scheme can be concretely instantiated using number-theoretic building blocks. We also propose a construction that relies on symmetric building blocks and prove its security (in the standard model) down to the assumption that PRGs exist.

8.4.1 Seekable Sequential Key Generators From Shortcut Permutations

We propose a generic construction of an SSKG from a shortcut one-way permutation and a cryptographic hash function, and prove that it achieves indistinguishability with forward security (IND-FS) in the random oracle model. We know from the previous section that ScPs exist and, in the case of our two instantiations, that are provably one way down to the hardness of the RSA and the SQRT problems. Thus, we obtain provably secure instantiations of SSKGs from random oracles and standard cryptographic assumptions.

Our construction, that we name *permute-then-hash*, essentially works as follows: the initial state is chosen by picking uniformly at random an element of the domain of the permutation; the next state is computed as the image of the current state under the permutation in use; each epoch's key is generated by applying the hash function to the state corresponding to that epoch; finally, the seeking procedure makes use of the permutation's Express algorithm. We validate the soundness of the *permute-then-hash* design by reducing its IND-FS security to the one-wayness of the ScP, in the random oracle model.

$\mathsf{Gen}(1^\lambda)$
01 $(P, sc) \leftarrow_\$ \mathsf{ScPGen}(1^\lambda)$
02 $(par, sk) \leftarrow (P, sc)$
03 Return (par, sk)

$\mathsf{Init}(par)$
04 $P \leftarrow par$
05 $x_0 \leftarrow_\$ D_P$
06 $st_0 \leftarrow (P, 0, x_0)$
07 Return st_0

$\mathsf{Evolve}(st_i)$
08 Parse st_i as (P, i, x_i)
09 $x_{i+1} \leftarrow \mathsf{Eval}(P, x_i)$
10 $st_{i+1} \leftarrow (P, i+1, x_{i+1})$
11 Return st_{i+1}

$\mathsf{GetKey}(st_i)$
12 Parse st_i as (P, i, x_i)
13 $K_i \leftarrow H(P, i, x_i)$
14 Return K_i

$\mathsf{Seek}(sk, st_0, m)$
15 $sc \leftarrow sk$
16 Parse st_0 as $(P, 0, x_0)$
17 $x_m \leftarrow \mathsf{Express}(sc, x_0, m)$
18 $st_m \leftarrow (P, m, x_m)$
19 Return st_m

Figure 8.2: *Specification of the* ScP-SSKG*'s algorithms (Construction 8).*

Construction 8 (ScP-SSKG)**.** *Let* $\mathsf{ScP} = (\mathsf{ScPGen}, \mathsf{Eval}, \mathsf{Express})$ *be a shortcut permutation, and let* $H\colon \{0,1\}^* \to \{0,1\}^{\ell(\lambda)}$ *be a hash function for some polynomial* ℓ*. Then the algorithms of our seekable sequential key generator* ScP-SSKG *are specified in Figure 8.2.*

Correctness of Construction 8 follows by inspection, while its security is stated in the next theorem. Note that ScP-SSKG supports, in principle, an unlimited number of epochs. However, as the theorem's bound shows, the higher T is, the larger the security loss. Thus, in practice it is reasonable to fix an upper bound for T which does not degrade the scheme's security.

Theorem 16 (Security of ScP-SSKG)**.** *Construction 8 offers IND-FS security in the random oracle model as long as* ScP *is a shortcut one-way permutation. More precisely, for every efficient adversary* $\mathcal{A}$ *against* ScP-SSKG *there exists an efficient adversary* $\mathcal{B}$ *against* ScP *such that*

$$\mathbf{Adv}^{\text{IND-FS}}_{\mathsf{ScP\text{-}SSKG},\mathcal{A}}(\lambda) \leq 2T \cdot \mathbf{Adv}^{\text{OWP}}_{\mathsf{ScP},\mathcal{B}}(\lambda) \ .$$

Proof. Let $\mathcal{A}$ be an adversary that plays the IND-FS game against the scheme. Recall that we are in the random oracle model, hence the adversary is given oracle access to the hash function H. Our security argument formalizes the intuition that $\mathcal{A}$ could tell apart a real key K_n from a random one only by querying state st_n to the random oracle (clearly, obtaining the value $H(st_n)$ discloses the nature, real or random, of the challenge key K_n^b with overwhelming probability). We show, however, that $\mathcal{A}$ asks such a hash query only with negligible probability. In what follows we formalize this intuition.

Let $\mathrm{E}^{0,b}_{\mathcal{A}}$ denote the experiment $\mathrm{Expt}^{\text{IND-FS},b}_{\mathsf{ScP\text{-}SSKG},\mathcal{A}}$. Consider an arbitrary execution of $\mathrm{E}^{0,b}_{\mathcal{A}}$. Let P be the public parameter generated for the shortcut permutation and denote the domain and associated bijection by D_P and $\pi\colon D_P \to D_P$, respectively. For the sake of legibility, throughout the proof we omit all occurrences of the parameter P. Accordingly, we denote the scheme's initial state by $st_0 = (0, x_0)$, further states by $st_t = (t, x_t)$ for $t \in \mathbb{N}$ and $x_t = \pi(x_{t-1}) = \pi^t(x_0)$, and keys by $K_t = H(t, x_t)$. We also ignore the maximum number of epochs T for now.

We proceed by defining a sequence of games that starts with $\mathrm{E}_{\mathcal{A}}^{0,b}$ and terminates with a game in which $\mathcal{A}$ has no advantage. In the first hop we define game $\mathrm{E}_{\mathcal{A}}^{1,b}$ from $\mathrm{E}_{\mathcal{A}}^{0,b}$ by forcing termination of the game with output 0 if $\mathcal{A}$ ever poses query (n, x_n) to H. Let bad^b be the event that $\mathcal{A}$ asks query (n, x_n) to the random oracle when the hidden bit is set to b. By definition, $\mathrm{E}_{\mathcal{A}}^{0,b}$ and $\mathrm{E}_{\mathcal{A}}^{1,b}$ are identical as long as bad^b does not occur. We can thus bound $\mathcal{A}$'s advantage in the IND-FS experiment as follows:

$$\begin{aligned}\mathbf{Adv}^{\text{IND-FS}}_{\mathsf{ScP\text{-}SSKG},\mathcal{A}}(\lambda) &= \left|\Pr\left[\mathrm{E}_{\mathcal{A}}^{0,1}\right] - \Pr\left[\mathrm{E}_{\mathcal{A}}^{0,0}\right]\right| \\ &\leq \left|\Pr\left[\mathrm{E}_{\mathcal{A}}^{0,1}\right] - \Pr\left[\mathrm{E}_{\mathcal{A}}^{1,1}\right]\right| + \left|\Pr\left[\mathrm{E}_{\mathcal{A}}^{1,1}\right] - \Pr\left[\mathrm{E}_{\mathcal{A}}^{1,0}\right]\right| + \left|\Pr\left[\mathrm{E}_{\mathcal{A}}^{1,0}\right] - \Pr\left[\mathrm{E}_{\mathcal{A}}^{0,0}\right]\right| \\ &\leq \Pr\left[bad^1\right] + \Pr\left[bad^0\right] + \left|\Pr\left[\mathrm{E}_{\mathcal{A}}^{1,1}\right] - \Pr\left[\mathrm{E}_{\mathcal{A}}^{1,0}\right]\right| \ .\end{aligned}$$

We will show later that both $\Pr\left[bad^0\right]$ and $\Pr\left[bad^1\right]$ can be upper bounded by the advantage of an efficient inverter against the underlying shortcut permutation ScP. Going on, we define the next game $\mathrm{E}_{\mathcal{A}}^{2,b}$ from $\mathrm{E}_{\mathcal{A}}^{1,b}$ by letting not only K_n^0 but also K_n^1 be randomly chosen. Observe that this transition is fictitious: deriving $K_n^1 \leftarrow H(n, x_n)$ from the random oracle or choosing it uniformly at random looks the same to the adversary as long as $\mathcal{A}$ does not ask the hash query (n, x_n). In fact, both $\mathrm{E}_{\mathcal{A}}^{1,b}$ and $\mathrm{E}_{\mathcal{A}}^{2,b}$ penalize the adversary in case it poses such query. Thus, $\mathcal{A}$'s advantage in game $\mathrm{E}_{\mathcal{A}}^{1,b}$ is the same as in game $\mathrm{E}_{\mathcal{A}}^{2,b}$. Observe finally that $\mathrm{E}_{\mathcal{A}}^{2,0}$ and $\mathrm{E}_{\mathcal{A}}^{2,1}$ are the very same game (K_n^0 and K_n^1 are both uniformly chosen), and hence in game $\mathrm{E}_{\mathcal{A}}^{2}$ the adversary has exactly zero advantage. The two observations above imply:

$$\left|\Pr\left[\mathrm{E}_{\mathcal{A}}^{1,1}\right] - \Pr\left[\mathrm{E}_{\mathcal{A}}^{1,0}\right]\right| = \left|\Pr\left[\mathrm{E}_{\mathcal{A}}^{2,1}\right] - \Pr\left[\mathrm{E}_{\mathcal{A}}^{2,0}\right]\right| = 0 \ ,$$

and lead us to the bound:

$$\mathbf{Adv}^{\text{IND-FS}}_{\mathsf{ScP\text{-}SSKG},\mathcal{A}}(\lambda) \leq \Pr\left[bad^0\right] + \Pr\left[bad^1\right] \ . \tag{8.1}$$

It remains to show that $\mathcal{A}$ poses the hash query (n, x_n) only with negligible probability, i.e., $\Pr[bad^b]$ is negligible for $b \in \{0, 1\}$. We prove a stronger statement, namely that all hash queries (t, x_t) with $0 \leq t < m$ occur with negligible probability. In the rest of the proof we refer to hash queries of the form (t, z) with $0 \leq t < m$ and $z = x_t$ as *lucky* queries, and we denote by $lucky^b$ the event that $\mathcal{A}$ poses a lucky query during an execution of experiment $\mathrm{E}_{\mathcal{A}}^{0,b}$. Clearly $\Pr[bad^b] \leq \Pr[lucky^b]$. We proceed with giving an explicit reduction $\mathcal{B}$ that uses $\mathcal{A}$ as a black box and succeeds in inverting the permutation ScP whenever $\mathcal{A}$ makes a lucky query to H, as stated in the following lemma.

Lemma 9. *Let $b \in \{0, 1\}$ and let $lucky^b$ denote the event that $\mathcal{A}$ poses a lucky query during the execution of experiment $\mathrm{E}_{\mathcal{A}}^{0,b}$. There exists an efficient algorithm $\mathcal{B}$ such that:*

$$\mathbf{Adv}^{\text{OWP}}_{\mathsf{ScP},\mathcal{B}}(\lambda) \geq \frac{1}{T} \cdot \Pr\left[lucky^b\right] \ . \tag{8.2}$$

Since $\Pr[bad^b] \leq \Pr[lucky^b]$, the inequalities (8.1) and (8.2) allow us to derive the final bound for $\mathcal{A}$'s IND-FS advantage:

$$\mathbf{Adv}^{\text{IND-FS}}_{\mathsf{ScP\text{-}SSKG},\mathcal{A}}(\lambda) \leq 2T \cdot \mathbf{Adv}^{\text{OWP}}_{\mathsf{ScP},\mathcal{B}}(\lambda) \ .$$

It remains to prove Lemma 9. We want to build an inverter $\mathcal{B}$ for π whose strategy relies on $\mathcal{A}$'s ability of making lucky queries. Assume that $\mathcal{B}$ receives a challenge $y \in D_P$ for the OWP game. Intuitively, the idea is to embed $\mathcal{B}$'s challenge into the state $st_m = (m, x_m)$, where m

$\mathcal{B}(1^\lambda, P, y)$:
01 $S_{keys} \leftarrow \emptyset, S_{hash} \leftarrow \emptyset$
02 $m^* \leftarrow_\$ [2\,..\,T-1]$
03 $(hist, n, m) \leftarrow_\$ \mathcal{A}_1^{\mathcal{O}_{\mathsf{GetKey}}(\cdot), H(\cdot)}(1^\lambda)$
04 Require $0 < n < m < T \ \wedge \ m^* = m$
05 $K_n^b \leftarrow_\$ \{0,1\}^{\ell(\lambda)}$
06 $st_m \leftarrow (m, y)$
07 $b' \leftarrow_\$ \mathcal{A}_2^{\mathcal{O}_{\mathsf{GetKey}}(\cdot), H(\cdot)}(hist, st_m, K_n^b)$
08 Return $\perp$

If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{GetKey}}(t)$:
09 Require $0 \le t < T$
10 If $\exists (t, K_t) \in S_{keys}$:
11 Return K_t to $\mathcal{A}$
12 Else:
13 $K_t \leftarrow_\$ \{0,1\}^{\ell(\lambda)}$
14 $S_{keys} \leftarrow S_{keys} \cup \{(t, K_t)\}$
15 If $t \ge m^*$:
16 $x_t \leftarrow \pi^{t-m^*}(y)$
17 $S_{hash} \leftarrow S_{hash} \cup \{((t, x_t), K_t)\}$
18 Return K_t to $\mathcal{A}$

If $\mathcal{A}$ queries $H(t, z)$:
19 If $\exists ((t, z), h) \in S_{hash}$:
20 Return h to $\mathcal{A}$
21 Else:
22 $h \leftarrow_\$ \{0,1\}^{\ell(\lambda)}$
23 $S_{hash} \leftarrow S_{hash} \cup \{((t, z), h)\}$
24 If $t < m^*$:
25 If $\pi^{m^*-t}(z) = y$: // lucky query?
26 $x \leftarrow \pi^{m^*-t-1}(z)$
27 Return x
28 Else if $\pi^{t-m^*}(y) = z$ // in-line query?
29 $S_{keys} \leftarrow S_{keys} \cup \{(t, h)\}$
30 Return h to $\mathcal{A}$

Figure 8.3: *Security reduction $\mathcal{B}$ described in the proof of Lemma 9. $\mathcal{B}$ receives an ScP challenge $(P, y) = (N, y)$ and embeds it into state $st_m = (N, m, x_m)$. It maintains sets S_{keys} and S_{hash} for recording the the values (t, K_t) and $((t, x_t), K_t)$ given to the adversary as $\mathcal{O}_{\mathsf{GetKey}}$ and H responses, respectively.*

is the corruption epoch that $\mathcal{A}$ requests, by setting $x_m \leftarrow y$ and hope that $\mathcal{A}$ poses a lucky query (t, x_t); if she does we have $t \le n < m$ and hence $\pi^{m-t}(x_t) = y$; then $\mathcal{B}$ outputs $\pi^{m-t-1}(x_t)$ as candidate preimage and, if *lucky*^{b} occurred, it wins the OWP game. The most challenging part of the proof is to guarantee that $\mathcal{B}$ provides a sound simulation of the IND-FS experiment for $\mathcal{A}$. Indeed, in the simulation, when processing queries to $\mathcal{O}_{\mathsf{GetKey}}$ and H we have to make sure that the two oracles answer consistently, i.e., that $\mathcal{O}_{\mathsf{GetKey}}(t) = H(q_t)$ for all queries $q_t = (t, x_t)$ that correspond to a valid state.

Consider an execution of the IND-FS game; we can assume without loss of generality that $\mathcal{A}$ makes only 'clever' queries to H, i.e., she tries to hit lucky queries by guessing pairs (t, z) with $t \in \mathbb{N}$ and $z \in D_P$. For any given initial state $st_0 = (0, x_0)$ we say that a hash query (t, z) is *in-line* if $t \in \mathbb{N}$ and $z = \pi^t(x_0)$; in this case we denote the value (t, z) by $q_t = (t, x_t)$. According to this notion, consistency between oracles H and $\mathcal{O}_{\mathsf{GetKey}}$ means that for every in-line query q_t it must be $H(q_t) = \mathcal{O}_{\mathsf{GetKey}}(t)$. Also note that a hash query (t, z) is lucky if it is in-line and $t < m$.

Importantly, $\mathcal{B}$ cannot check whether $\mathcal{A}$'s queries are in-line, i.e., whether they hit any valid state, until $\mathcal{A}$ declares the epoch m to be corrupted. In fact, from $\mathcal{B}$'s perspective the value x_0 is implicitly defined by the equality $y = \pi^m(x_0)$ and, in particular, it is determined only when both y and m are specified. So, how can $\mathcal{B}$ ensure consistency of $\mathcal{O}_{\mathsf{GetKey}}$ and H's answers if it cannot detect which queries are in-line? We overcome the issue by letting $\mathcal{B}$ guess the value of m and then define in-line queries with respect to its guess and the challenge value y it receives.

In more detail, $\mathcal{B}$ starts by choosing m^* uniformly at random from $[2\,..\,T-1]$. It then simulates the oracle $\mathcal{O}_{\mathsf{GetKey}}$ by returning randomly chosen ℓ-bit strings; in case $\mathcal{A}_1$ requests a key K_t for $t \geq m^*$, $\mathcal{B}$ must also assure consistency with the hash value $H(q_t)$ for the corresponding in-line (but *not* lucky) query $q_t = (t, x_t)$; it does so by programming the oracle H. Similarly, $\mathcal{B}$ replies to hash queries with randomly chosen ℓ-bit strings. However, this time it has to identify in-line queries to ensure consistency with the answers of $\mathcal{O}_{\mathsf{GetKey}}$. To this end, it deems a query (t, z) as a candidate in-line query by checking value z against y as follows. Assuming that $\mathcal{B}$ correctly guessed $m^* = m$, for $t \geq m^*$ we have that query (t, z) is in-line if and only if $z = \pi^{t-m^*}(y)$. Similarly, for $t < m^*$, query (t, z) is in-line if and only if $y = \pi^{m^*-t}(z)$; note that in this case (t, z) is, in particular, a lucky query. Given this, $\mathcal{B}$'s strategy is as follows: it assigns a fresh hash value h to every candidate in-line query (t, z) and register key $K_t := h$; moreover, if $t < m^*$ then $\mathcal{B}$ detected a candidate lucky query and extracts from it the candidate preimage $x := \pi^{m^*-t-1}(z)$ of y. Eventually $\mathcal{A}_1$ declares challenge epoch n and corruption epoch m: now $\mathcal{B}$ can check whether its guess on m was correct and, if not, it terminates the simulation; otherwise it returns a randomly chosen ℓ-bit string K_n^b together with a 'state' (m, y) to $\mathcal{A}_2$, hence it proceeds by answering further queries as before until either a lucky query is posed—in this case $\mathcal{B}$ will for sure invert y successfully—or $\mathcal{A}_2$ stops. We give an explicit description of the reduction $\mathcal{B}$ in Figure 8.3.

Observe that in the first phase of the simulation, before $\mathcal{A}$ declares the corruption epoch m, $\mathcal{B}$ provides a perfect simulation of the oracles $\mathcal{O}_{\mathsf{GetKey}}$ and H and, once m has been announced, the simulation proceeds only if $m = m^*$. In case $\mathcal{B}$ goes on with the second phase, its simulation of the oracles is perfect as long as $\mathcal{A}$ does not ask a lucky query. However, if $\mathcal{A}$ does ask a lucky query then $\mathcal{B}$ successfully extracts a preimage of y. We can finally relate $\mathcal{B}$'s inverting advantage with the probability that event lucky^b happens:

$$\mathbf{Adv}^{\mathrm{OWP}}_{\mathcal{B},\mathsf{ScP}}(\lambda) = \Pr\left[m^* = m \ \wedge \ \mathit{lucky}^b\right] = \frac{1}{T-2} \cdot \Pr\left[\mathit{lucky}^b\right]$$

from which the claimed bound immediately follows. □

8.4.2 Seekable Sequential Key Generators From Pseudorandom Generators

We have seen in the previous sections how to generically construct seekable sequential key generators from any shortcut one-way permutation. The concrete ScPs considered in Section 8.4.1 are given by the squaring operation modulo a Blum integer N, respectively, the exponentiation to the eth power modulo N in an RSA setting, while applying the shortcut algorithm corresponds to reducing a certain exponent modulo $\varphi(N)$. Under the assumption that the hash function in use is a random oracle, the forward security of the resulting SSKG is implied by the one-wayness of its underlying ScP, and its seekability is based on the ScP's shortcut property. A notable technical artifact of the squaring-based ScP is that seekability requires knowledge of $\varphi(N)$ while forward security requires this value to be unknown. This dilemma is side-stepped by giving only the owners of a *seeking key* (e.g., a log auditor in the logging scenario) the ability to fast-forward through the SSKG output sequence while denying this functionality to the users. However, the necessity of this extra key should be considered an artifact of the number-theory-based constructions FACT-ScP and RSA-ScP from Section 8.3: there, the seeking key contains the factorization of the modulus N underlying the schemes: as the proposed Evolve algorithm is one way only if this factorization is not known, the Seek algorithm is available exclusively to those who know the seeking key as a 'trapdoor'.

A natural question arises: Do shortcut permutations exist that need no additional information other than $x \in D_P$ and $k \in \mathbb{N}$ to compute $\pi^k(x)$ in sublinear time? Here we take a different approach and ask ourselves a more general question: Do SSKGs exist that require

no seeking key to perform the Seek operation? We provide a positive answer by providing an explicit construction of what we call a *seeking-key-free* SSKG. Formally, we say that an SSKG is seeking-key-free if the seeking algorithm needs no seeking key sk and, by consequence, we also require that for every security parameter $\lambda \in \mathbb{N}$, every number of supported epochs $T \in \mathbb{N}$, and every pair $(par, sk) \leftarrow_{\$} \mathsf{Gen}(1^\lambda, T)$ it holds $sk = \varepsilon$.

In this section we show how to build seeking-key-free seekable sequential key generators. The scheme that we propose here relies only on symmetric building blocks, and can thus be instantiated using PRGs, block ciphers, or hash functions. Moreover, the scheme enjoys a security proof in the standard model. Before giving the details of our construction we recall, and set the notation for, stacks and binary trees (these are the non-cryptographic building blocks of our scheme).

Stacks and their operations. A *stack* is a standard data structure for the storage of objects. Stacks follow the last-in first-out principle: the last element stored in a stack is the first element to be read back (and removed). The following procedures can be used to operate on stacks for storing, reading, and deleting elements. By $\mathsf{Init}(S)$ we denote the initialization of a fresh and empty stack S. To add an element x 'on top of' stack S we use the operation $\mathsf{Push}(S, x)$. We write $x \leftarrow \mathsf{Pop}(S)$ for reading and removing the top element of S. Finally, with $x \leftarrow \mathsf{Peek}_k(S)$ the k-th element of S can be read without deleting it; here, elements are counted from the top, i.e., $\mathsf{Peek}_1(S)$ reads the top-most element. When using these notation, the operations Init, Push, and Pop are understood to modify their argument S in place, while Peek_k leaves it unchanged.

Binary and d-ary trees. A *tree* is a simple, undirected, connected graph without cycles. We particularly consider rooted trees, i.e., trees with a distinguished *root* node. The nodes adjacent to the root node are called its *children*; each child can be considered, in turn, the root of a subtree. The *level* L of a node indicates its distance (in nodes) to the root, where we assign level $L = 1$ to the latter. Children of the same node are *siblings* of each other. We will assume that the children of each node are ordered, i.e., can be identified by a number $1 \le i \le d$, where d is the number of children. For two siblings with indices i and j, respectively, in case $i < j$ we say that node i is *left* of node j and that node j is *right* of node i. In binary trees we may also refer to the children as left and right directly. Nodes that have no children are called *leaves*, all other nodes are called *internal*. A tree is *d-regular* (or *d-ary*, or *binary* in case $d = 2$) if every internal node has exactly d children. We focus on d-ary trees of constant height h, i.e., where all leaves have the same level $L = h$. If $\mu_d(L)$ denotes the number of nodes at level L, then for such trees we have $\mu_d(1) = 1$ and $\mu_d(L) = d \cdot \mu_d(L-1) = d^{L-1}$ for all $L > 1$. For the total number of nodes $\nu_d(H)$ we hence obtain

$$\nu_d(h) = \sum_{L=1}^{h} \mu_d(L) = \sum_{L=1}^{h} d^{L-1} = (d^h - 1)/(d-1) \ ,$$

by the geometric summation formula. As a special case, for binary trees the total number of nodes is $\nu_2(h) = 2^h - 1$. We finally define the notion of *co-path* of a node. Let v denote an arbitrary node of a tree. Intuitively speaking, the (right) co-path of v is the vector of the right siblings of the nodes on the (unique) path connecting the root node with v; if for individual nodes on this path there are multiple right siblings, all of them appear in the co-path. For a formal definition, let L denote the level of $v = v_L$ and let $(v_1, \ldots, v_L)$ denote the path that connects the root (denoted here with v_1) with v_L. For each $1 \le i \le L$ let $\mathbf{V}_r(v_i)$ be the vector of right siblings of node v_i, in left-to-right order (some of these vectors might be empty, and particularly $\mathbf{V}_r(v_1)$ always is). We define the co-path of v_L to be the vector $\mathbf{V}_r(v_L) \,\|\, \cdots \,\|\, \mathbf{V}_r(v_1)$ obtained by combining these vectors into a single one using concatenation.

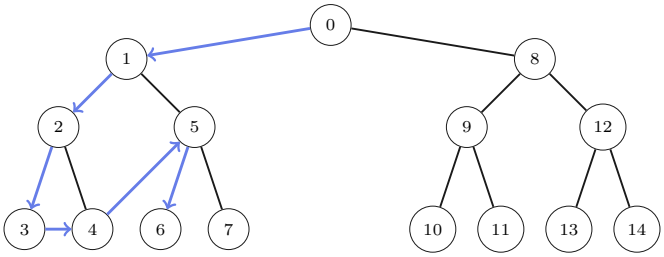


Figure 8.4: *A binary tree with height $H = 4$ and $N = 2^4 - 1 = 15$ nodes. The latter are numbered according to a pre-order depth-first search, as partially indicated by the arrow from the root node w_0 to node w_6.*

In our SSKG we identify time epochs with the nodes of a binary tree. More precisely, let $h \in \mathbb{N}$ denote the height of a binary tree and let $N = 2^h - 1$ denote the number of nodes of this tree. Given the pre-order depth-first enumeration $w_0, \ldots, w_{N-1}$ of the nodes (first visit the root, then recursively the left subtree, then recursively the right subtree, as illustrated in Figure 8.4), we let time epoch i and node w_i correspond.

The idea is to assign to each node w_i a (secret) seed $s_i \in \{0,1\}^\lambda$ from which the epoch's key K_i and the seeds of all subordinate nodes can be deterministically derived via PRG invocations. Here, exclusively the secret of the root node is assigned at random. Intuitively, the pseudorandomness of the PRG ensures that all keys and seeds look random to the adversary.

We proceed with specifying which information the states associated with the epochs shall record. Recall that from each state st_i, $0 \leq i < N$, two pieces of information have to be derivable: the epoch-specific key K_i and the successor state st_{i+1} (and, by induction, also all following states and keys). Clearly, in this construction, the notions of 'seed' and 'state' do not coincide; for instance, in the tree of Figure 8.4, key K_9 cannot be computed from just seed s_4. However, if state st_4 contained (s_4, s_5, s_8), then for all $4 \leq i < N$ the keys K_i could be computed from this state. Inspired by this observation, we let our SSKG store in each state st_i a collection of seeds, namely the seeds of the roots of the 'remaining subtrees'. The latter set of nodes is precisely the *co-path* of node w_i. Intuitively speaking, this construction is forward-secure as each state stores only the minimal information required to compute all succeeding states. In particular, as each node precedes all vertices on its co-path (in the sense of a pre-order visit of the tree), the key associated to that node remains secure even if any subsequent epoch's seed is leaked to the adversary.

We present next the algorithms of our SSKG construction. Particularly interesting, we believe, are the details on how the required pre-order depth-first search is implicitly performed by help of a stack data structure.

Construction 9 (TreeSSKG)**.** *Let $\ell\colon \mathbb{N} \to \mathbb{N}$ be a positive polynomial and let G be a pseudorandom generator with $|G(x)| = 2|x| + \ell(|x|)$ for all $x \in \{0,1\}^*$. For $s \in \{0,1\}^\lambda$ write*

$$G(s) = G_L(s) \,\|\, G_R(s) \,\|\, G_K(s) \quad \textit{where} \quad G_L(s), G_R(s) \in \{0,1\}^\lambda \textit{ and } G_K(s) \in \{0,1\}^{\ell(\lambda)} \ .$$

The algorithms of $\mathsf{TreeSSKG} = (\mathsf{Gen}, \mathsf{Init}, \mathsf{Evolve}, \mathsf{GetKey}, \mathsf{Seek})$ *are defined in Figure 8.5.*

Let us discuss the algorithms of TreeSSKG in greater detail.

Gen. Given the security parameter 1^λ and the desired number of epochs T, this algorithm computes the minimum number $H \in \mathbb{N}$ such that the binary tree of constant height H consists of at least T nodes. Observe that this tree may have more than T nodes, i.e., more epochs are

$\mathsf{Gen}(1^\lambda, T)$
01 $H \leftarrow \lceil \log(T+1) \rceil$
02 $N \leftarrow 2^H - 1$
03 $par \leftarrow (\lambda, N, H)$
04 Return par

$\mathsf{Init}(par)$
05 parse par as (λ, N, H)
06 $s \leftarrow_\$ \{0,1\}^\lambda$
07 $\mathsf{Init}(S)$
08 $\mathsf{Push}(S, (s, H))$
09 $st_0 \leftarrow S$
10 Return st_0

$\mathsf{Evolve}(st_i)$
11 $S \leftarrow st_i$
12 $(s, h) \leftarrow \mathsf{Pop}(S)$
13 If $h > 1$:
14 $\quad \mathsf{Push}(S, (G_R(s), h-1))$
15 $\quad \mathsf{Push}(S, (G_L(s), h-1))$
16 $st_{i+1} \leftarrow S$
17 Return st_{i+1}

$\mathsf{GetKey}(st_i)$
18 $S \leftarrow st_i$
19 $(s, h) \leftarrow \mathsf{Peek}_1(S)$
20 $K_i \leftarrow G_K(s)$
21 Return K_i

$\mathsf{Seek}(st_0, k)$
22 $S \leftarrow st_0$
23 $\delta \leftarrow k$
24 $(s, h) \leftarrow \mathsf{Pop}(S)$
25 While $\delta > 0$:
26 $\quad h \leftarrow h - 1$
27 $\quad$ If $\delta < 2^h$:
28 $\quad\quad \mathsf{Push}(S, (G_R(s), h))$
29 $\quad\quad s \leftarrow G_L(s)$
30 $\quad\quad \delta \leftarrow \delta - 1$
31 Else:
32 $\quad\quad s \leftarrow G_R(s)$
33 $\quad\quad \delta \leftarrow \delta - 2^h$
34 $\mathsf{Push}(S, (s, h))$
35 $st_k \leftarrow S$
36 Return st_k

Figure 8.5: *Specification of the* TreeSSKG*'s algorithms for binary trees (Construction 9). We assume G to be a PRG and use the symbols G_L, G_R, G_K to denote the output parts of G, as specified in Construction 9. Here H and N denote the height and the number of nodes, respectively, of the binary tree associated to the SSKG instance. Observe that the number of actually supported epochs is potentially larger than the number of requested epochs, i.e., $N \geq T$, due to the rounding operation in line 01 of* Gen*. The variable δ used within* Seek *represents the number of nodes that still have to be skipped.*

supported than required. Note that in contrast to the algorithm Gen from Construction 6, here the parameter generation algorithm is deterministic.

Init. After extracting from the public parameters the height $h = H$ of the underlying tree, this algorithm picks a random seed $s = s_0$ for the root node and stores in state st_0 a stack S that contains only a single element: the pair (s, h). Here and in the following, such pairs should be understood as 'seed s shall generate a subtree of height h'.

Evolve. The stack S stored in state st_i generally contains two types of information: the top element is a pair (s, h) associated with the current node w_i, and the remaining elements are associated with the corresponding pairs of the nodes on w_i's co-path. After taking the current entry (s, h) off the stack, in order to implement the depth-first search idea illustrated above, this algorithm distinguishes two cases: if node w_i is an internal node (i.e., $h > 1$), the update step computes the seeds of its two child nodes by invoking the underlying PRG, starting with the right seed as it needs to be prepended to the current co-path. The new seeds $G_L(s)$ and $G_R(s)$ can be considered roots of subtrees of one level less than w_i; they are hence pushed onto the

stack with decreased h-value. In the second case, if the current node w_i is a leaf (i.e., $h = 1$), no further action has to be taken: the next required seed is the 'left-most' node on w_i's co-path, which resides on the stack's top position already.

GetKey. Deriving the current key is particularly simple as it only requires to read the top-most element (s, h) of the stack $S = st_i$ and evaluate $G_K(s)$. Observe that the Peek_1 operation leaves its argument unchanged.

Seek. Deriving state st_k from the initial state st_0 via iteratively evoking k times the Evolve procedure is equivalent to visiting all nodes of the tree according to a pre-order traversal until reaching node w_k. However, there is an appealing way to obtain seed s_k more directly, without passing through all the intermediate vertices. The idea is to just walk down the path connecting the root node with w_k. Taking this shortcut decreases the seeking cost to only $\mathcal{O}(\log N)$, as opposed to $\mathcal{O}(N)$. This is the intuition behind the design of our algorithm Seek from Figure 8.5.

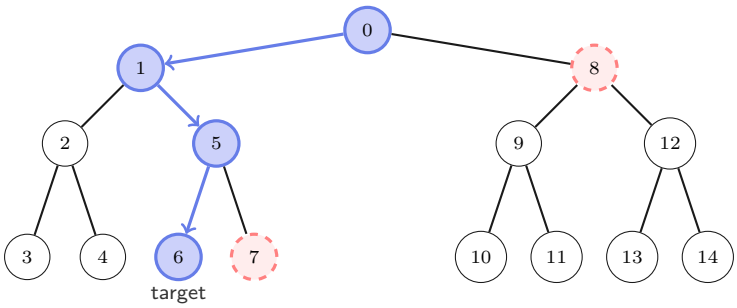


Figure 8.6: *A visualization of the procedure* Seek *when computing state* st_6. *As indicated by the arrows, the algorithm walks down the path from the root node* w_0 *to the target node* w_6 *(thick nodes); simultaneously, it records the nodes of* w_6*'s co-path, i.e.,* (w_7, w_8) *(dashed nodes).*

Recall that Seek is required to output the whole state st_k, and not just seed s_k. In other words, the execution of the algorithm needs to comprehend the construction of the co-path of node w_k. We provide details on how Seek fulfills this task. Our strategy, illustrated in Figure 8.6, is to walk down the path from the root to node w_k, recording the right siblings of the visited nodes on a stack. During this process, with a variable δ we keep track of the remaining number of epochs that needs to be skipped. This counter is particularly helpful for deciding whether, in the path towards w_k, the left or the right child node have to be taken. Indeed, the number of nodes covered by the left and right subtrees is $2^h - 1$ each; if $\delta \leq 2^h - 1$ then the left child is the next to consider, but the right child has to be recorded for the co-path. On the other hand, if $\delta \geq 2^h$, then the left child can be ignored, the co-path doesn't have to be extended, and the walk towards w_k is continued via the right child. The procedure terminates when for the number of remaining epochs we have $\delta = 0$, which means that we reached the target node w_k.

We now formally assess the security of Construction 9.

Theorem 17 (Security of Construction 9)**.** *Fix a number of epochs* T. *Let* $N \in N$ *and let* G *be a PRG as indicated in Construction 9. For every efficient adversary* $\mathcal{A}$ *against* TreeSSKG *there exists an efficient adversary* $\mathcal{B}$ *against* G *such that*

$$\mathbf{Adv}^{\text{IND-FS}}_{\mathsf{TreeSSKG},\mathcal{A}}(\lambda) \leq 2\log(N+1) \cdot \mathbf{Adv}^{\text{PRG}}_{G,\mathcal{B}}(\lambda) \ .$$

Proof. The security argument for our scheme reflects the intuition that every key K_i, for being (part of) the output of a PRG invocation, looks like a random string to any efficient adversary

as long as the seed used to compute it remains hidden. In the IND-FS experiment, in addition to state st_m, the adversary gets a challenge K_n^b, which is either the real key K_n in case $b = 1$, or it is a randomly chosen string of length $\ell(\lambda)$ otherwise; here, n and m are adversarially chosen conditioned on $n < m < N$. The state st_m reveals seed s_m and possibly some subsequent seeds. However, by construction, from these seeds none of the preceding states can be computed. Thus, corrupting state st_m should be of no help to the adversary in distinguishing keys prior to epoch m. In particular, key K_n can be expected to stay secure. We formalize this intuition in the following.

We make use of game hops which progressively transform the IND-FS experiment, denoted here by $\mathrm{E}_{\mathcal{A}}^{0,b}$, into one for which all adversaries have advantage exactly zero. In the following we use the shortcut $\Pr[\mathrm{E}_{\mathcal{A}}^{i,b}]$ to indicate the probability $\Pr[\mathrm{E}_{\mathcal{A}}^{i,b}(1^\lambda) = 1]$. Observe that in game $\mathrm{E}_{\mathcal{A}}^{0,b}$ key K_n^1 is not just computed as the output of a PRG on input a random seed; rather, it is computed by iterating a PRG up to $\log(N+1)$ times, where only the first input (seed s_0) is truly random. We hence proceed via a hybrid argument, by considering intermediate experiments $\mathrm{H}_{\mathcal{A}}^{0,b}, \mathrm{H}_{\mathcal{A}}^{1,b}, \ldots, \mathrm{H}_{\mathcal{A}}^{L,b}$, where we set $\mathrm{H}_{\mathcal{A}}^{0,b} = \mathrm{E}_{\mathcal{A}}^{0,b}$ where L denote the level of challenge node w_n (i.e., the node associated with the challenge epoch) in the tree. The idea is to let each transition from hybrid $\mathrm{H}_{\mathcal{A}}^{i-1,b}$ to hybrid $\mathrm{H}_{\mathcal{A}}^{i,b}$ replace the output of the PRG associated with the i-th node w_i on the path from the root node w_0 to node w_n with a value chosen uniformly at random. Then, in $\mathrm{H}_{\mathcal{A}}^{L,b}$ the challenge key K_n^b will be random independently of bit b: in case $b = 1$ due to the argument just given, and in case $b = 0$ by the definition of experiment $\mathrm{H}_{\mathcal{A}}^{L,0}$. In particular, every adversary $\mathcal{A}$ playing in $\mathrm{H}_{\mathcal{A}}^{L,b}$ will have zero advantage.

More precisely, let L be the level of node w_n and let $(v_1, \ldots, v_L)$ denote the path from the root $v_1 = w_0$ to node $v_L = w_n$. For every $i = 1, \ldots, L$, derive $\mathrm{H}_{\mathcal{A}}^{i,b}$ from $\mathrm{H}_{\mathcal{A}}^{i-1,b}$ by replacing the output of $G(s_k)$ with a random string in $\{0,1\}^{2\lambda+\ell}$, where k is the epoch number corresponding to node v_i.

Observe that, except for $\mathrm{H}^{0,b}$, as we follow the path top-to-bottom, seed s_k was replaced by a random value in the hybrid before, i.e., in $\mathrm{H}_{\mathcal{A}}^{i-1,b}$. By consequence, for every $\mathcal{A}$ there exists a distinguisher $\mathcal{B}_i$ whose advantage is at least the difference in probability between $\mathrm{H}_{\mathcal{A}}^{i-1,b}$ and $\mathrm{H}_{\mathcal{A}}^{i,b}$ as follows:

$$\left|\Pr\left[\mathrm{H}_{\mathcal{A}}^{i-1,b}\right] - \Pr\left[\mathrm{H}_{\mathcal{A}}^{i,b}\right]\right| \leq \mathbf{Adv}_{G,\mathcal{B}_i}^{\mathrm{PRG}}(\lambda) \ . \tag{8.3}$$

As already stated, the challenge key K_n^b in hybrid $\mathrm{H}_{\mathcal{A}}^{L,b}$ is uniformly random, independently of bit b. In other words, $\mathrm{H}_{\mathcal{A}}^{L,0}$ and $\mathrm{H}_{\mathcal{A}}^{L,1}$ are the very same experiment, and we have, even for unbounded distinguishers,

$$\left|\Pr\left[\mathrm{H}_{\mathcal{A}}^{L,1}\right] - \Pr\left[\mathrm{H}_{\mathcal{A}}^{L,0}\right]\right| = 0 \ . \tag{8.4}$$

Using an induction argument and the triangle inequality, we can combine (8.3) and (8.4) into

$$\left|\Pr\left[\mathrm{E}_{\mathcal{A}}^{1,1}\right] - \Pr\left[\mathrm{E}_{\mathcal{A}}^{1,0}\right]\right| \leq 2\sum_{i=1}^{L} \mathbf{Adv}_{G,\mathcal{B}_i}^{\mathrm{PRG}}(\lambda) \ . \tag{8.5}$$

Finally, let $\mathcal{B}$ be the algorithm that chooses $i \in [1\,..\,L]$ uniformly at random and run executes $\mathcal{B}_i$: then $\mathbf{Adv}_{G,\mathcal{B}}^{\mathrm{PRG}}(\lambda) = \frac{1}{L}\sum_{i=1}^{L} \mathbf{Adv}_{G,\mathcal{B}_i}^{\mathrm{PRG}}(\lambda)$. By plugging $\mathcal{B}$'s advantage in equation (8.5) and observing that $L \leq \log(N+1)$ we obtain the final bound. □

Extending TreeSSKG Towards d-ary Trees. We discuss how to extend our binary-tree-based construction towards the general case of d-ary trees, for arbitrary $d \geq 2$. Recall that a d-ary tree of constant height H has $\nu_d(H) = (d^H - 1)/(d-1)$ nodes. The intuition behind our design is mainly the same as in the binary case: we consider an enumeration $w_0, \ldots, w_{N-1}$ of

the tree's nodes according to a pre-order depth-first search (first visit the root, then, from left to right, recursively the subtrees) and associate with every node w_i a seed s_i from which epoch's key K_i and, where applicable, the seeds of its children are derived using a PRG. It is clear that this PRG must have a larger expansion than that of Construction 9: every PRG invocation has to yield one (sub)string of length $\ell(\lambda)$ for the key, and d-many (sub)strings of length λ for the subordinate seeds. We specify the algorithms below.

Construction 10 ($\mathsf{TreeSSKG}$ for d-ary trees)**.** *Let $\ell\colon \mathbb{N} \to \mathbb{N}$ be a positive polynomial, let $d \in \mathbb{N}^{\geq 2}$, and let G be a PRG such that $|G(x)| = d \cdot |x| + \ell(|x|)$ for all $x \in \{0,1\}^*$. For $s \in \{0,1\}^\lambda$ write*

$$G(s) = G_1(s) \,\|\, \cdots \,\|\, G_d(s) \,\|\, G_K(s) \quad \textit{where} \quad G_j(s) \in \{0,1\}^\lambda \; \forall j \in [1\,..\,d] \textit{ and } G_K(s) \in \{0,1\}^{\ell(\lambda)} \ .$$

Our SSKG based on d-ary trees $\mathsf{TreeSSKG_d} = (\mathsf{Gen}_d, \mathsf{Init}_d, \mathsf{Evolve}_d, \mathsf{GetKey}_d, \mathsf{Seek}_d)$ is defined by the algorithms in Figure 8.7.

$\mathsf{Gen}_d(1^\lambda, T)$
01 $H \leftarrow \lceil \log_d(T+1) \rceil$
02 $N \leftarrow (d^H - 1)/(d-1)$
03 $par \leftarrow (\lambda, N, H)$
04 Return par

$\mathsf{Init}_d(par)$
05 Parse par as (λ, N, H)
06 $s \leftarrow_\$ \{0,1\}^\lambda$
07 $\mathsf{Init}(S)$
08 $\mathsf{Push}(S, (s, H))$
09 $st_0 \leftarrow S$
10 Return st_0

$\mathsf{Evolve}_d(st_i)$
11 $S \leftarrow st_i$
12 $(s, h) \leftarrow \mathsf{Pop}(S)$
13 If $h > 1$:
14 For $j = d$ down to 1:
15 $\mathsf{Push}(S, (G_j(s), h-1))$
16 $st_{i+1} \leftarrow S$
17 Return st_{i+1}

$\mathsf{GetKey}_d(st_i)$
18 $S \leftarrow st_i$
19 $(s, h) \leftarrow \mathsf{Peek}_1(S)$
20 $K_i \leftarrow G_K(s)$
21 Return K_i

$\mathsf{Seek}_d(st_0, k)$
22 $S \leftarrow st_0$
23 $\delta \leftarrow k$
24 $(s, h) \leftarrow \mathsf{Pop}(S)$
25 While $\delta > 0$:
26 $h \leftarrow h - 1$
27 $\nu \leftarrow (d^h - 1)/(d-1)$
28 $c \leftarrow \lfloor (\delta - 1)/\nu \rfloor + 1$
29 For $j = d$ down to $c+1$:
30 $\mathsf{Push}(S, (G_j(s), h))$
31 $s \leftarrow G_c(s)$
32 $\delta \leftarrow \delta - (1 + (c-1)\nu)$
33 $\mathsf{Push}(S, (s, h))$
34 $st_k \leftarrow S$
35 Return st_k

Figure 8.7: *Algorithms of $\mathsf{TreeSSKG_d}$ (for d-ary trees). We assume G to be a PRG and use the symbols G_j, $j \in [1\,..\,d]$ and G_K to denote the output parts of G, as specified in Construction 10. The integers H and N denote the height, respectively, the number of nodes, of the binary tree associated to the SSKG instance. Observe that the number of actually supported epochs is potentially larger than the number of requested epochs, i.e., $N \geq T$, due to the rounding operation in line 01 of Gen_d. The variable δ used within Seek_d represents the number of nodes that still have to be skipped.*

Observe that the proposed Init_d and GetKey_d algorithms are identical with the ones from the binary case, and that the only modification in Gen_d is the basis to which the logarithm is taken for computing H. More interesting is the new Evolve_d algorithm. Here, whenever an internal

node needs to be expanded, all seeds of the d direct successors are computed; the left-most seed will be associated with the next state (i.e., st_{i+1}), and its right siblings become part of the new co-path, i.e., are pushed in the correct order onto the stack. We point out two differences in the Seek_d algorithm. Firstly, in line 28, with c we calculate the number $1 \leq c \leq d$ of current node's child that is on the path from the root to target node w_k (consequently, only the siblings with numbers $c+1, \ldots, d$ need to be recorded for the co-path). Secondly, in line 32, the number δ of epochs that still need to be skipped is decreased by $(c-1)\nu$ in one shot, where ν indicates the number of nodes of the subtree generated by the respectively considered seed s.

We give the following security statement for our generalized construction. The proof is essentially the same to the one for Theorem 17 (it suffices to replace the underlying binary tree with a d-ary tree).

Theorem 18 (Security of $\mathsf{TreeSSKG_d}$)**.** *Fix a number of epochs T. Let $N \in N$ and let G be a PRG as indicated in Construction 10. For every efficient adversary $\mathcal{A}$ against $\mathsf{TreeSSKG}_d$ there exists an efficient adversary $\mathcal{B}$ against G such that*

$$\mathbf{Adv}^{\text{IND-FS}}_{\mathsf{TreeSSKG_d},\mathcal{A}}(\lambda) \leq 2\log_d(N+1) \cdot \mathbf{Adv}^{\text{PRG}}_{G,\mathcal{B}}(\lambda) \ .$$

Remark 11 (Degenerate trees)*.* In the above definitions we insisted on fixing d such that $d \geq 2$. This choice guarantees that the time it takes to seek to an arbitrary target node is $\mathcal{O}(\log_d N)$. Observe however, that also for $d = 1$ our scheme is correct and secure, but falls back to linear seeking time. It is interesting to observe that this degenerate case corresponds exactly with the hash chain approach of early (not seekable) SKGs constructions (e.g., from [KS98, SK99, BY03]).

8.4.3 Enhancing the Seeking Functionality

As required by the notion of seekability the Seek algorithm allows computing any state st_k given the initial state st_0. Observe, however, that for some applications this initial state might not be accessible; indeed, forward security can be attained only if states of expired epochs are securely erased. From a practical perspective it is hence appealing to generalize the functionality of Seek to allow efficient computation of st_{i+k} from any state st_i, and not just from st_0, thus realizing the Evolve^k functionality for arbitrary starting points.

Note that for SSKGs based on shortcut permutations, such a flexible functionality is always available: if one wishes to compute state $st_j = (P, j, x_j)$ from state $st_i = (P, i, x_i)$, $i < j$, it suffices to invoke the ScP algorithm $\mathsf{Express}$ on input the shortcut sc, starting point x_i, and number of epochs to be skipped $k = j - i$. Enhancing the seeking procedure of our PRG-based construction requires, instead, a little tweak that we discuss next.

Assume that a $\mathsf{TreeSSKG}$ instance is in state st_i and an application requests it to seek to position st_{i+k}, for arbitrary $0 \leq i \leq i+k < N$. Recall from the discussion in Section 8.4.2 that state st_i encodes both the seed s_i and the co-path of node w_i. Recall also that, as a property of the employed pre-order visit of the tree, for each state st_j with $j > i$, the co-path of node w_i contains an ancestor w of w_j. Following these observations, our tweaked Seek construction consists of two consecutive phases. For seeking to state st_{i+k}, in the first phase the algorithm considers all nodes on the co-path of w_i until it finds the ancestor w of w_{i+k}. The second phase is then a descent from that node to node w_{i+k}, similarly to what we had in the regular Seek algorithms. In both phases care has to be taken that the co-path of the target node w_{i+k} is correctly assembled as part of st_{i+k}. The working principle of our new seeking method is also illustrated in Figure 8.8.

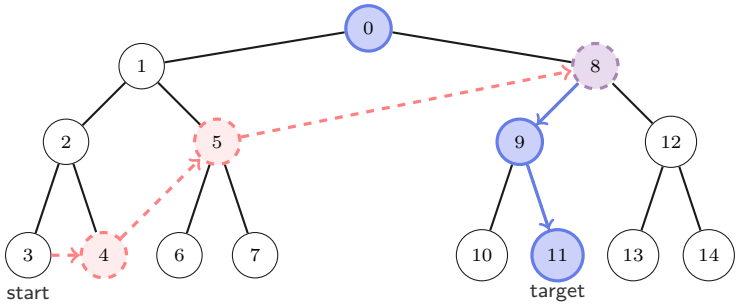


Figure 8.8: *A visualization of the enhanced seeking procedure of* TreeSSKG*, jumping from epoch* 3 *to epoch* 11*. As indicated by the arrows, the algorithm first finds the intersection, here* w_8*, between the co-path of the start node* w_3 *(dashed nodes) and the path that connects the root with the target node* w_{11} *(thick nodes); from there it proceeds downwards until it reaches target node* w_{11}*.*

8.5 Practical aspects and deployment

The SSKG algorithms from Constructions 8 and 9, with minor changes for optimization purposes, have been implemented. An experimental analysis of the performance of the relative implementations can be found in [MP14]. Here we summarize the main results.

8.5.1 Sequential key generators based on shortcut permutations

An implementation of the SSKG from Construction 8 that uses FACT-ScP as underlying permutation (named FACT-SSKG in the present section) is available [Poe] and was contributed by the authors of [MP13]. In fact, the code became part of the *journald* logging component of the *systemd* system and service manager, a core piece of virtually all current Linux-based distribution [sys]. The resulting SSKG is used in combination with a cryptographic MAC in order to implement secured local logging, called *Forward-Secure Sealing* in journald. Generation of initial state st_0 takes place on the system whose logs are to be protected. Each time the SSKG state is evolved, a MAC tag protecting the data written since the previous MAC operation is appended to the log file. Forward-Secure Sealing provides an offline verification tool that checks the MAC tag sequence of log files. If a log file is manipulated, the verification tool will determine the time range where the integrity of the log file is intact. When the SSKG state is evolved, particular care is taken to ensure that the previous state is securely deleted from the file system and underlying physical storage. In order to optimize the storage size of the initial generator's state as well as the seeking time, the implementation slightly deviates from the construction presented in this thesis. Recall that in a logging scenario the initial state st_0 is first created by the log auditor using Gen and then distributed to all hosts to be monitored. When FACT-ScP is used, between 1024 to 4096 bits would have to be copied, depending on the desired level of security [BCC+12], just counting the size of $x_0 \in \mathbb{QR}_N$. For the journald implementation, the Gen algorithm is provided with an explicit random seed of short length (e.g., 80–128 bits), and it generates the initial state using a PRG. Using this tweak, only 128 bits (or less) have to be shared among the hosts. The second modification improves the efficiency of the Express operation by using a standard trick [JK03, BS02] to speed up private operations in factoring-based schemes via the Chinese Remainder Theorem (CRT). For instance, if an exponentiation $y \leftarrow x^k \bmod N$ is to be computed and the factorization $N = pq$ is known, then y can be obtained by CRT-decomposing x into $x_p \leftarrow x \bmod p$ and $x_q \leftarrow x \bmod q$, by computing $y_p \leftarrow x_p^{k \bmod \varphi(p)} \bmod p$ and $y_q \leftarrow x_q^{k \bmod \varphi(q)} \bmod q$ independently of each other, and by mapping (y_p, y_q) back to $\mathbb{Z}_N$ (by applying the CRT a second time). The described method

to compute x^k is approximately four times faster than evaluating the term directly, without the CRT [MvV97, Note 14.75].

8.5.2 Sequential key generators from pseudorandom generators

In practice, PRGs can be instantiated using blockciphers, stream ciphers, or hash functions. For instance, a blockcipher operated in counter mode can be seen as a PRG where the block cipher's key acts as the PRG's seed. Similar counter-based constructions derived from hash functions or PRFs (e.g., HMAC) are possible. Our scheme TreeSSKG was implemented by the authors of [MP14], both in the setting of binary trees and of d-ary trees, considering four different PRG instantiations that rely on the AES128 and AES256 block ciphers and the MD5 and SHA256 hash functions. That is, we have two instantiations at the $\lambda = 128$ security level, and two at the $\lambda = 256$ level. The performance of such implementation was experimentally evaluated in [MP14] using the following setup. After generating TreeSSKG instances with parameters $d = 2$ and $H = 20$ (i.e., supporting $N = 2^{20} - 1 \approx 10^6$ epochs), the Evolve algorithm was iterated through all epochs in linear order to determine both the the average and the worst-case time. Similar measures were performed for Gen and GetKey (here, average and worst-case coincide), and for the average and worst-case time it takes for the Seek algorithm to recover states st_k, ranging over all values $k \in [0, N-1]$.

The results of the performance analysis from [MP14] are summarized in Table 8.1. We point out that for FACT-SSKG, the analogue of Gen in [MP13] in fact consists of two separate algorithms: one that produces public parameters and an associated seeking key, and one that generates the actual initial SSKG state. As any fixed combination of public parameters and corresponding seeking key can be used for many SSKG instances without security compromises, for fairness the comparison does not count the generation costs of the former when indicating the Gen performance in Table 8.1.

It is instructive to also study the required state sizes for both TreeSSKG and FACT-SSKG. In the TreeSSKG implementation, for fixed parameter d, the (maximum) state size scales roughly linearly in both H and the seed length of the used PRG. Concretely, for $(d, H) = (2, 20)$ and 128 bit keys (e.g., for AES128- and MD5-based PRGs) the state requires 350 bytes, while for 256 bit security a total of 670 bytes of storage are necessary. In the FACT-SSKG scheme the space in the state variable is taken by a modulus N, a value $x \in \mathbb{Z}_N^*$, a 64 bit epoch counter, and a small header. Precisely, for 2048 and 3072 bit moduli this results in 522 and 778 bytes of state, respectively.

8.5.3 Results and discussion

We discuss the results from Table 8.1, beginning with those of TreeSSKG (i.e., columns AES128, MD5, AES256, and SHA256). Our first observation is that the Gen time is independent of the respectively used PRG. This is not surprising as the former algorithm never invokes the latter, but spends its time with memory allocation and requesting the random starting seed from OpenSSL's core routines. The timings for Evolve indicate that, as expected, 128-bit cryptographic primitives are faster than 256-bit primitives, and that for a fixed security level the hash-function-based constructions are (slightly) preferable. The hypothesis that the time spent by the individual algorithms is dominated by the internal PRG executions is supported by the observation that the running time of Evolve (on average) and GetKey coincide, and that the worst-case running time of Evolve is twice that value; to see this, recall that Evolve executions perform either two internal PRG invocations or none, and that the average number of invocations is one.

	AES128		**MD5**		**SQRT/2048 bit**
	[average]	[max]	[average]	[max]	
Gen	22μs		22μs		27μs
Evolve	0.2μs	0.5μs	0.2μs	0.4μs	8μs
GetKey	0.2μs		0.2μs		12μs
Seek	7μs	9μs	6μs	7μs	4.9ms
	AES256		**SHA256**		**SQRT**/3072 bit
	[average]	[max]	[average]	[max]	
Gen	22μs		22μs		38μs
Evolve	0.5μs	1μs	0.4μs	0.8μs	13μs
GetKey	0.4μs		0.4μs		13μs
Seek	14μs	18μs	11μs	15μs	12.6ms

Table 8.1: Efficiency measurements for the TreeSSKG algorithms (instantiated with different PRGs) and for the FACT-SSKG algorithms. All experiments were performed on an Intel Core i7-3517U CPU clocked at 1.90GHz. The implementations rely on OpenSSL version 0.9.8 for TreeSSKG and on the gcrypt library in version 1.5.0 for FACT-SSKG.

The routines of FACT-SSKG are clearly outperformed by the ones of TreeSSKG. Firstly, for the tree-based Evolve algorithm the timing values are about 30 times better than those for the factoring-based algorithm (recall that the latter's state update involves a modular squaring operation). Similar results show the tree-based GetKey algorithm to be faster, by a factor between 30 and 60, depending on the considered security level. This might be surprising at first sight, as the factoring-based GetKey consists of just hashing the corresponding state variable, but presumably the explication for this difference is in the considerably larger state sizes. Finally, the superiority of the TreeSSKG construction in terms of efficiency is made even more evident by studying the performance of the seeking algorithms, which show the tree-based seeking procedure being 700–1000 times faster than the corresponding factoring-based procedure, again depending on the security level.

Chapter 9

Conclusion and Open Problems

In this thesis we reconsider the common approach of modeling secure channels as (stateful) authenticated encryption (AE) primitives. Essentially such models assume that a sender transmits encrypted plaintexts to a receiver, and that an attacker controlling the network may observe and actively tamper with the sent ciphertexts, replay or reorder them, or inject its own. Concerning this approach we identify three aspects that we consider extremely important for real-world channel protocols but were not addressed in the literature so far.

In particular, cryptographic models, with the exception of [BDPS12], assume that messages and ciphertexts transmitted from the sender to the receiver are atomic units and, thus, are processed as such by the channel algorithms. Inspired by [BDPS12], which introduces the notion of symmetric encryption supporting ciphertext fragmentation at the receiver, we note that transport layer security protocols such as TLS and SSH also allow for fragmentation at the sender and, moreover, seem to be specifically designed for transporting a stream of data rather than atomic messages. To narrow the gap between these real-world channel protocols and our theoretical understanding of them we initiate the study of stream-based channels. For these protocols we propose novel syntax and functionality that captures a stream-oriented API, formalize the security properties that we expect from a stream-based channel, and demonstrate their feasibility by providing a natural construction that is close in spirit to a recent design of the TLS Record Protocol.

Because of this enriched functionality of stream-based channels, our confidentiality experiments employ a rather complex suppression mechanism in the definition of the receiving oracle. One may wonder if this level of complexity is really necessary. When formalizing functionality and security, we followed the rationale of capturing a wide range of stream-oriented behavior with a single channel notion. It is plausible that by restricting the stream-based channel functionality one may obtain a simplified model that is easier to understand and work with, yet remains relevant and expressive. Defining a restricted functionality that is still rich enough to describe real-world channel protocols remains an open issue. We also leave open how to modify the confidentiality experiment against active attacks in a way that the scheme presented in Section 3.3.3, which is intuitively confidential but declared insecure within our model, would be indeed considered secure. The fact that this scheme, which does not provide integrity, is deemed insecure in our model highlights that our confidentiality notion is conservative and, to some extent, has some form of integrity built in.

Looking at secure channels from a different perspective, we note that while cryptographic models account for unidirectional communication, from one sender to one receiver, in practice channel protocols are typically employed for bidirectional communication. Further, some applications like multi-party chat protocols involving more than two communicating parties need interaction in multiple directions. The inherent interactiveness of a multi-directed, multi-user

setting adds a new dimension of complexity to the communication, and leaves space for attacks exploiting falsified causal dependencies between exchanged messages that do not apply to the simplified unidirected setting. To understand what security means in such a setting we introduce new channel notions that extend the stateful AE abstraction to allow for multiple participants and for interactive communication. These channels provide extended functionality and strictly stronger security properties than unidirectional channels. Nevertheless, they can be constructed from standard, symmetric cryptographic building blocks, as we show.

Our model for broadcast communication assumes that the set of participants is fixed in advance. An interesting open problem is to generalize our notions to allow for dynamic groups, i.e., letting participants join and leave the group at any point in time. Here we see as a main challenge here to define the concept of causal past (or history), given that the causal property, as it is, cannot be fulfilled any longer.

We finally note that in the theoretical analysis of secure channel protocols forward security does not seem to be an explicit objective while it generally is for authenticated key exchange protocols. Modern secure channels like TLS 1.3 and instant messaging protocols, however, list (not always explicitly) forward security as one of their goals. It is considered folklore that the forward-secure variant of an AE scheme can be bootstrapped from a traditional AE scheme by refreshing its key using a forward-secure key generation mechanism. Aiming at modular constructions of forward-secure channel protocols we focus on building schemes that generate a sequence of keys with forward security. For such schemes we introduce the novel functionality of seekability, which essentially gives random access to the keys of the sequence, and propose constructions of seekable key generators that enjoy (forward) security from standard cryptographic building blocks.

Bibliography

[APW09] Martin R. Albrecht, Kenneth G. Paterson, and Gaven J. Watson. Plaintext recovery attacks against SSH. In *2009 IEEE Symposium on Security and Privacy*, pages 16–26, Oakland, CA, USA, May 17–20, 2009. IEEE Computer Society Press.

[AW04] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics (2nd edition)*. John Wiley Interscience, March 2004.

[BCC+12] Steve Babbage, Dario Catalano, Carlos Cid, Benne de Weger, Orr Dunkelman, Christian Gehrmann, Louis Granboulan, Tim Güneysu, Jens Hermans, Tanja Lange, Arjen Lenstra, Chris Mitchell, Mats Näslund, Phong Nguyen, Christof Paar, Kenny Paterson, Jan Pelzl, Thomas Pornin, Bart Preneel, Christian Rechberger, Vincent Rijmen, Matt Robshaw, Andy Rupp, Martin Schläffer, Serge Vaudenay, Fré Vercauteren, and Michael Ward. ECRYPT Yearly Report on Algorithms and Keysizes, September 2012. `http://www.ecrypt.eu.org/documents/D.SPA.20.pdf`.

[BMM+15] Christian Badertscher, Christian Matt, Ueli Maurer, Phillip Rogaway, and Björn Tackmann. Augmented secure channels and the goal of the TLS 1.3 record layer. In Man Ho Au and Atsuko Miyaji, editors, *ProvSec 2015: 9th International Conference on Provable Security*, volume 9451 of *Lecture Notes in Computer Science*, pages 85–104, Kanazawa, Japan, November 24–26, 2015. Springer, Heidelberg, Germany.

[BPS15a] Guy Barwell, Dan Page, and Martijn Stam. Rogue decryption failures: Reconciling AE robustness notions. Cryptology ePrint Archive, Report 2015/895, 2015. `http://eprint.iacr.org/2015/895`.

[BPS15b] Guy Barwell, Daniel Page, and Martijn Stam. Rogue decryption failures: Reconciling AE robustness notions. In Jens Groth, editor, *15th IMA International Conference on Cryptography and Coding*, volume 9496 of *Lecture Notes in Computer Science*, pages 94–111, Oxford, UK, December 15–17, 2015. Springer, Heidelberg, Germany.

[BDJR97] Mihir Bellare, Anand Desai, Eric Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *38th Annual Symposium on Foundations of Computer Science*, pages 394–403, Miami Beach, Florida, October 19–22, 1997. IEEE Computer Society Press.

[BKN02] Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. Authenticated encryption in SSH: Provably fixing the SSH binary packet protocol. In Vijayalakshmi Atluri, editor, *ACM CCS 02: 9th Conference on Computer and Communications Security*, pages 1–11, Washington D.C., USA, November 18–22, 2002. ACM Press.

[BM99] Mihir Bellare and Sara K. Miner. A forward-secure digital signature scheme. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666

of *Lecture Notes in Computer Science*, pages 431–448, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Heidelberg, Germany.

[BN00] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Tatsuaki Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545, Kyoto, Japan, December 3–7, 2000. Springer, Heidelberg, Germany.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press.

[BY97] Mihir Bellare and Bennet S. Yee. Forward integrity for secure audit logs. Technical report, Department of Computer Science and Engineering, University of California at San Diego, 1997.

[BY03] Mihir Bellare and Bennet S. Yee. Forward-security in private-key cryptography. In Marc Joye, editor, *Topics in Cryptology – CT-RSA 2003*, volume 2612 of *Lecture Notes in Computer Science*, pages 1–18, San Francisco, CA, USA, April 13–17, 2003. Springer, Heidelberg, Germany.

[BDPS12] Alexandra Boldyreva, Jean Paul Degabriele, Kenneth G. Paterson, and Martijn Stam. Security of symmetric encryption in the presence of ciphertext fragmentation. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 682–699, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

[BDPS14] Alexandra Boldyreva, Jean Paul Degabriele, Kenneth G. Paterson, and Martijn Stam. On symmetric encryption with distinguishable decryption failures. In Shiho Moriai, editor, *Fast Software Encryption – FSE 2013*, volume 8424 of *Lecture Notes in Computer Science*, pages 367–390, Singapore, March 11–13, 2014. Springer, Heidelberg, Germany.

[BS02] Dan Boneh and Hovav Shacham. Fast variants of RSA. *RSA Cryptobytes*, 5(1):1–9, Winter/Spring 2002.

[BSWW13] Christina Brzuska, Nigel P. Smart, Bogdan Warinschi, and Gaven J. Watson. An analysis of the EMV channel establishment protocol. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13: 20th Conference on Computer and Communications Security*, pages 373–386, Berlin, Germany, November 4–8, 2013. ACM Press.

[CGR11] Christian Cachin, Rachid Guerraoui, and Luís Rodrigues. *Introduction to Reliable and Secure Distributed Programming (2. ed.)*. Springer, 2011.

[CKPS01] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 524–541, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.

[CHK07] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. *Journal of Cryptology*, 20(3):265–294, July 2007.

[CK01] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474, Innsbruck, Austria, May 6–10, 2001. Springer, Heidelberg, Germany.

[CK02] Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 337–351, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Heidelberg, Germany.

[CKN03] Ran Canetti, Hugo Krawczyk, and Jesper Buus Nielsen. Relaxing chosen-ciphertext security. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 565–582, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany.

[CHVV03] Brice Canvel, Alain P. Hiltgen, Serge Vaudenay, and Martin Vuagnoux. Password interception in a SSL/TLS channel. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 583–599, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany.

[DP10] Jean Paul Degabriele and Kenneth G. Paterson. On the (in)security of IPsec in MAC-then-encrypt configurations. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 10: 17th Conference on Computer and Communications Security*, pages 493–504, Chicago, Illinois, USA, October 4–8, 2010. ACM Press.

[DR08] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176.

[FGMP15] Marc Fischlin, Felix Günther, Giorgia Azzurra Marson, and Kenneth G. Paterson. Data is a stream: Security of stream-based channels. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 545–564, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

[GUVC09] Ian Goldberg, Berkant Ustaoglu, Matthew Van Gundy, and Hao Chen. Multi-party off-the-record messaging. In Ehab Al-Shaer, Somesh Jha, and Angelos D. Keromytis, editors, *ACM CCS 09: 16th Conference on Computer and Communications Security*, pages 358–368, Chicago, Illinois, USA, November 9–13, 2009. ACM Press.

[Gut96] Peter Gutmann. Secure deletion of data from magnetic and solid-state memory. In *Proceedings of the Sixth USENIX Security Symposium, San Jose, CA*, volume 14, 1996.

[HLT03] M. Jason Hinek, Mo King Low, and Edlyn Teske. On some attacks on multi-prime RSA. In Kaisa Nyberg and Howard M. Heys, editors, *SAC 2002: 9th Annual International Workshop on Selected Areas in Cryptography*, volume 2595 of *Lecture*

Notes in Computer Science, pages 385–404, St. John's, Newfoundland, Canada, August 15–16, 2003. Springer, Heidelberg, Germany.

[JKSS12] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DHE in the standard model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 273–293, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.

[JK03] J. Jonsson and B. Kaliski. Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1. RFC 3447 (Informational), February 2003.

[KL15] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography.* Cryptography and Network Security Series. Chapman and Hall/CRC Press, 2015.

[KCC10] J. Kelsey, J. Callas, and A. Clemm. Signed Syslog Messages. RFC 5848 (Proposed Standard), May 2010.

[KS98] John Kelsey and Bruce Schneier. Cryptographic support for secure logs on untrusted machines. In *Proceedings of the 7th USENIX Security Symposium*, 1998.

[KS99] John Kelsey and Bruce Schneier. Minimizing bandwidth for remote access to cryptographically protected audit logs. In *Recent Advances in Intrusion Detection*, 1999.

[KPB03] Tadayoshi Kohno, Adriana Palacio, and John Black. Building secure cryptographic transforms, or how to encrypt and mac. Cryptology ePrint Archive, Report 2003/177, 2003.

[Kra01] Hugo Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is SSL?). In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 310–331, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.

[KPW13] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. On the security of the TLS protocol: A systematic analysis. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 429–448, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

[Lam78] Leslie Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM*, 21(7):558–565, July 1978.

[MP13] Giorgia Azzurra Marson and Bertram Poettering. Practical secure logging: Seekable sequential key generators. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013: 18th European Symposium on Research in Computer Security*, volume 8134 of *Lecture Notes in Computer Science*, pages 111–128, Egham, UK, September 9–13, 2013. Springer, Heidelberg, Germany.

[MP14] Giorgia Azzurra Marson and Bertram Poettering. Even more practical secure logging: Tree-based seekable sequential key generators. In Miroslaw Kutylowski and Jaideep Vaidya, editors, *ESORICS 2014: 19th European Symposium on Research in Computer Security, Part II*, volume 8713 of *Lecture Notes in Computer Science*, pages 37–54, Wroclaw, Poland, September 7–11, 2014. Springer, Heidelberg, Germany.

[MP17] Giorgia Azzurra Marson and Bertram Poettering. Security notions for bidirectional channels. *IACR Transactions on Symmetric Cryptology*, 2017. (To appear).

[MRT12] Ueli Maurer, Andreas Rüedlinger, and Björn Tackmann. Confidentiality and integrity: A constructive perspective. In Ronald Cramer, editor, *TCC 2012: 9th Theory of Cryptography Conference*, volume 7194 of *Lecture Notes in Computer Science*, pages 209–229, Taormina, Sicily, Italy, March 19–21, 2012. Springer, Heidelberg, Germany.

[MvV97] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. The CRC Press series on discrete mathematics and its applications. CRC Press, 2000 N.W. Corporate Blvd., Boca Raton, FL 33431-9868, USA, 1997.

[Nam02] Chanathip Namprempre. Secure channels based on authenticated encryption schemes: A simple characterization. In Yuliang Zheng, editor, *Advances in Cryptology – ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 515–532, Queenstown, New Zealand, December 1–5, 2002. Springer, Heidelberg, Germany.

[OTR16] Off-the-Record Messaging. `http://otr.cypherpunks.ca`, 2016.

[PRS11] Kenneth G. Paterson, Thomas Ristenpart, and Thomas Shrimpton. Tag size does matter: Attacks and proofs for the TLS record protocol. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 372–389, Seoul, South Korea, December 4–8, 2011. Springer, Heidelberg, Germany.

[PW10] Kenneth G. Paterson and Gaven J. Watson. Plaintext-dependent decryption: A formal security treatment of SSH-CTR. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 345–361, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany.

[PW01] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *2001 IEEE Symposium on Security and Privacy*, pages 184–200, Oakland, CA, USA, May 2001. IEEE Computer Society Press.

[Poe] Bertram Poettering. `fsprg` – seekable forward-secure pseudorandom generator. `http://cgit.freedesktop.org/systemd/systemd/tree/src/journal/fsprg.c`.

[Pos81] J. Postel. Transmission Control Protocol. RFC 793 (INTERNET STANDARD), September 1981. Updated by RFCs 1122, 3168, 6093, 6528.

[RB94] Michael K. Reiter and Kenneth P. Birman. How to securely replicate services. *ACM Transactions on Programming Languages and Systems*, 16(3):986–1009, 1994.

[RG95] Michael K. Reiter and Li Gong. Securing causal relationships in distributed systems. *Comput. J.*, 38(8):633–642, 1995.

[Res16] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3 – draft-ietf-tls-tls13-15. `http://tools.ietf.org/html/draft-ietf-tls-tls13-15`, August 2016.

[Rog02] Phillip Rogaway. Authenticated-encryption with associated-data. In Vijayalakshmi Atluri, editor, *ACM CCS 02: 9th Conference on Computer and Communications Security*, pages 98–107, Washington D.C., USA, November 18–22, 2002. ACM Press.

[SK99] Bruce Schneier and John Kelsey. Secure audit logs to support computer forensics. *ACM Trans. Inf. Syst. Secur.*, 2(2):159–176, 1999.

[SM94] Reinhard Schwarz and Friedemann Mattern. Detecting causal relationships in distributed computations: In search of the holy grail. *Distributed Computing*, 7(3):149–174, 1994.

[Sho99] Victor Shoup. On formal models for secure key exchange. Technical Report RZ 3120, IBM, 1999.

[sys] `systemd`: System and service manager. `http://www.freedesktop.org/wiki/Software/systemd/`.

[Tex14] TextSecure. `http://whispersystems.org`, 2014.

[YL06] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Protocol Architecture. RFC 4251 (Proposed Standard), January 2006.