Interpretation of Provable Security for Cryptographic Practice

Christopher Patton

PhD Dissertation
Submitted June 2020

Acknowledgements

This work was made possible by those who provided guidance, support, and technical feedback throughout the process of carrying it out. There are too many of you to thank by name, but I'd like to call out those of you who contributed ideas to this thesis.

First, thanks to Tom Shrimpton, my advisor, mentor, and friend, who taught me how to think like a cryptographer and identify problems worth solving. The hours we spent in his office staring at a chalkboard are among the most intellectually satisfying moments of my career. Thank you as well to Kevin Butler, Patrick Traynor, Domenic Forte, and countless other faculty and students at FICS for helping to clarify the role of my work in the broader context of computer security.

A special thanks to Trevor Perrin, whose feedback on our original analysis of Noise (CRYPTO 2019 [117]) lead to a much more general result. Our lengthy discussions over the summer of 2019 helped to clarify my thinking on this and a variety of other topics relevant to this dissertation. Thanks also to Mihir Bellare, who in shepherding our submission to CCS 2018 [116], helped shape some of our formalisms. Thanks to Nick Sullivan, who hosted my internship at Cloudflare during the summer of 2018, which provided practical experience that inspired many of the problems tackled in this work. A huge thanks to Phil Rogaway, who introduced me to the wonderful world of cryptography and instilled in me the values that every researcher and practitioner should carry.

Finally, thank you to all of you who have provided less tangible, but no less vital support over the years. I especially want to thank my brother David Patton, who was the first to show me computers for what they are (and that being a nerd is cool). Thank you to my parents, the rest of my brothers and sisters, and my friends for molding me into the free-thinking, inquisitive, and hard-working person that I am. Last but certainly not least, I thank Elizabeth King for her unshakable support, her strength, and most of all her thoughtfulness. From the bottom of my heart, all the way to the top—thank you.

For my friend Julia_.

Abstract

The provable security methodology is an effective tool for ruling out attacks against cryptographic systems. Yet for a given system, it succeeds only to the degree to which the system's formal specification faithfully captures its observed behavior. Practical considerations regarding the standardization, implementation, and deployment of cryptography regularly create discrepancies between the real system and the subject of the analysis, resulting in a gap between what is known about the system and its security in the real world. This dissertation offers a critical perspective on the task of reconciling provable security with the translation of cryptography into practice. Its main contribution is a formal characterization of translations that are "safe", in the sense that it is possible to patch the existing proof in order to account for the observed behavior. Based on this characterization, we develop an analytical framework in which security of the translated system is proved by appealing to results already established for the original. This strategy preserves the rigor of a direct proof, but bares a significantly lower cost in terms of analytical effort. We demonstrate this by exhibiting a number of exercises and case studies that cover a wide range of problems. The diversity of these problems illustrates the framework's utility as a guide for interpreting provable security in practice.

Contents

Chapter 1

Introduction

There are a handful of problems with which mathematicians and computer scientists have grappled with for decades, and sometimes centuries, to no avail. For example:

Problem 1. Given an integer $N > 1$, find primes $p_1, p_2, \ldots, p_e$ such that $N = p_1 \cdot p_2 \cdots p_e$.

Problem 2. Given a prime $p > 2$, the generator g of a multiplicative group of integers modulo p, and a point X in the group, find x such that $X = g^x$.

Problem 3. Given the basis of an n-dimensional lattice $\mathcal{L} \subset \mathbb{R}^n$, find a shortest non-zero vector $\vec{v} \in \mathcal{L}$.

These problems and others like them are "hard" in the sense that no (classical) algorithm is known for solving them efficiently. Remarkably, the premise of secure communication in the modern world is that mounting a successful attack is as hard as solving these kinds of problems. Establishing this premise is the principle task of cryptography and its discipline of *provable security.*

Provable security concerns the formal modeling and analysis of computer systems, especially those that use cryptography in order to meet their goals. The task begins by specifying an *experiment* that models the scheme's execution environment (i.e., how the adversary interacts with it) and the attacker's objective. We say the scheme is secure if every efficient (i.e., polynomial-time) adversary's probability of success is small (i.e., negligible in the "security parameter"). This can be shown by exhibiting a *reduction* from some hard problem to the scheme's security: given an efficient attacker A that breaks Π with probability δ, the idea is to construct an efficient algorithm B^A that solves some problem P with probability (close to) δ; if this can be done, then P being hard implies that Π is secure.

The potential for this methodology to rule out attacks has profoundly transformed the practice of cryptography [132], so much so that a security reduction is often a prerequisite for the standardization, implementation, and eventual deployment of cryptographic systems. Yet it is common for these very same processes to create discrepancies between the real system and its formal specification. The subject of this dissertation is the task of reconciling these discrepancies with existing analysis.

1.1 Translation and the Limits of Provable Security

In this work, the term *system* refers to both a cryptographic scheme and the environment in which it is executed. As a concrete example, think of a set of clients and servers communicating with one another over the Internet using the TLS[1] protocol [123]. A system's provable security treatment (e.g., Bhargavan

[1] Transport Layer Security.

et al. for TLS [39]) necessarily includes a formal specification of the scheme and its execution environment; and when discrepancies arise between the real system and its formal specification, it is necessary to revise the specification to account for the observed behavior. In this thesis, we refer to this process as *translation*.

Translation is often intentional, as by the standardization of a protocol in the literature (this process often requires making changes in order to account for operational details not reflected in the original scheme) or by the addition of a new feature into an existing standard (e.g., via a protocol extension). It can also be the unintentional (or unavoidable) consequence of deployment or implementation, as by the (accidental) reuse of the system's secret keys in other applications. In general, a translation is a change to a scheme's specification or environment that creates a gap between what is known about the system and its security in practice. Because this gap can result in attacks, it has catalyzed the development of proof techniques for modeling as much real-world behavior as possible.

1.1.1 A Priori Analysis

Due to its crucial role in our daily lives, much of this effort has focused on the TLS protocol, standardized by RFCs[2] 8446 (version 1.3), 5246 (1.2), 4346 (1.1), and 2246 (1.0). Each of these documents is the culmination of a community-driven process that aims to balance the wide ranging and often conflicting interests of organizations who hold a stake in the protocol's design. Rather than fully specify a single protocol, TLS admits a variety of valid implementations, each having its own set of features and security properties. The protocol has been in wide use for more than 20 years (the first version was adopted in 1999 [11]), and in that time, it has undergone significant evolution. Until recently, its development was driven primarily by the discovery of attacks and responses to those attacks [113]: only for the latest version (adopted in 2018 [123]) was provable security pursued for the complete standard, prior to the protocol's deployment.

As TLS has evolved, so have our analytical tools. The earliest formal treatments for the protocol [85, 91, 96] considered feature sets in isolation (i.e., a single choice of authentication mechanism, protocol version, ciphersuite, set of extensions, etc.), but overtime, various breakthroughs enabled analysis in increasingly sophisticated detail, including modeling the mechanism whereby the client and server choose (i.e., negotiate) the set of features to use [43, 100, 41]. Among the most innovative approaches were the mechanization of provable security, as embodied by the miTLS project [42, 45, 67]; and the "Partially Specified Protocols" (PSP) framework of Rogaway and Stegers [130].

Mechanization. What distinguishes a mechanized proof from a conventional "pen-and-paper" one is that the system is specified in a programming language with a formal semantics that lends itself to the study of cryptographic security goals.[3] This enables the use of a theorem prover to help find a proof; and once a proof is found, its verification is completely automated, thereby increasing our confidence in the proof's correctness. Mechanization provides several advantages, not the least of which is that it eases the analytical cost of making revisions in light of changes to the protocol's specification [39]. This feature allows the analysis to scale to complex systems like TLS.

Despite this advantage, existing mechanized proofs of TLS share one feature with their pen-and-paper counterparts that, ultimately, limits their ability to account for the full range of real-world behavior. An important feature of TLS, and other protocols of similar complexity, is that its standard only partially specifies the behavior of the communicants such that the security provided by any given implementation depends upon how it interprets unspecified details. Existing mechanized proofs for TLS apply only to a single implementation of the standard.

[2] Request For Comments.

[3] A number of languages have been developed for this purpose, e.g., CryptoVerif [46]. Related languages like Tamarin [19] and ProVerif [47] are only suitable for Dolev-Yao-style symbolic analysis [69]: in this dissertation we are interested in standard, computational security properties.

The PSP Methodology. A work by Rogaway and Stegers from 2009 [130] provides an elegant solution to this problem. When devising the protocol's formal specification, their strategy is to divide the protocol into two components: the "protocol core (PC)", comprised of the functionalities that are essential to security; and the "specification details (SD)", which captures everything else. Formally, the SD is thought of as an oracle that the PC makes calls to in order to implement the full protocol. But in the security experiment, SD-oracle queries are answered by the adversary, thereby formalizing a kind of worst-case behavior of the SD: intuitively, if security can be proven, then the protocol is secure no matter how the SD is realized. This is an exceptionally strong attack model, but one that yields a rigorous treatment of the standard itself, and not just a single interpretation of its unspecified behavior. In theory, it allows the analyst to specify the (potentially large) set of behaviors the real system might exhibit without impacting the analysis, i.e., without the need to revise the formal specification.

Both of these methodologies make significant strides, but the benefits they provide for practice are essentially at the limit of what a priori security analysis can do. The state of provable security of TLS is a patchwork of results that, together, paint a fairly complete picture of the protocol's real-world security. And this analytical effort has paid off: as our tools have evolved, so have avenues of attack become more sophisticated and harder to exploit [10, 134]. But the TLS standard is subject to continual change, as new use cases and operational requirements continue to drive the protocol's development. As a result of these forces, ensuring that its security rests on firm, formal foundations remains an on-going challenge.

1.1.2 A Posteriori Analysis

When considering a change to a cryptographic protocol or its execution environment, the immediate task is to determine whether the translation leads to an attack. Assuming the original system is supported by a proof of security, this analysis requires an intimate understanding of the existing proof in order to decide if it can be "patched" to account for the change. Furthermore, preserving the same level of rigor for the translated system as for the original entails generating a fresh proof (i.e., "applying the patch"). This task bares a significant analytical cost, one that makes a rigorous analysis prohibitive. Especially when the change is (or at least appears to be) relatively straightforward, it is common to give an informal argument that sketches how to patch the original proof (e.g., [30, §5], [59, §9], and [93, §5]). Thus, the central question of this dissertation is whether the rigor of the existing analysis can be preserved without resorting to a fresh proof.

The answer we provide is that, for many translations, it is possible to prove security by appealing directly to the existing result. In particular, we prove a composition lemma that, for "safe" translations, yields a reduction from the security of the original system to the security of the translated system, thereby allowing one to argue security by reasoning about the translation itself. The centerpiece of this dissertation is its *translation framework* (presented in Chapter 2), which provides a formal characterization of not only safe translations, but also those that are "risky", in the sense that they could lead to an attack. We provide an overview of the framework in the next section.

1.2 The Translation Framework

Our framework begins with a new look at an old idea. In particular, we extend the notion of *indifferentiability* of Maurer, Renner, and Holenstein [104] (hereafter MRH) to the study of cryptographic protocols. Indifferentiability has become an important tool for provable security. Most famously, it provides a precise way to argue that security in the random oracle model (ROM) [29] is preserved when the random oracle (RO) is instantiated by a concrete hash function that uses a "smaller" idealized primitive, such as a compression function modeled as an RO. Coron et al. [60] were the first to explore this application of indifferentiability,

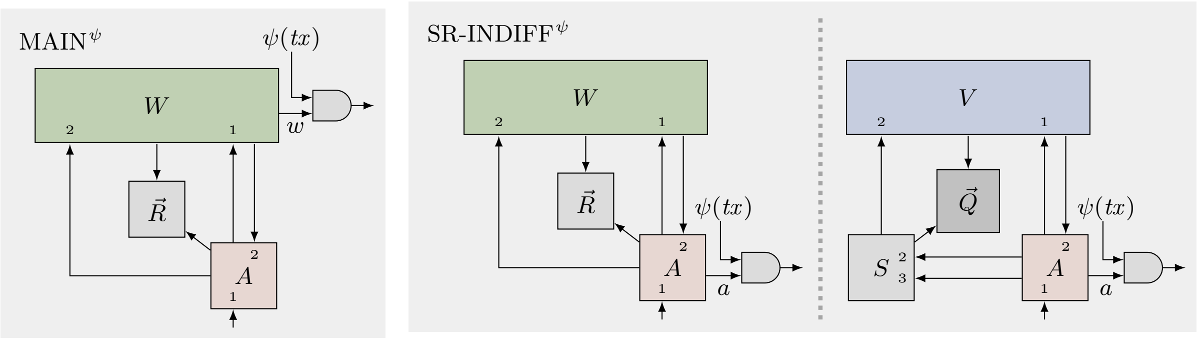

Figure 1.1: Illustration of the MAIN$^{\psi}$ (Def. 1) and SR-INDIFF$^{\psi}$ (Def. 3) security experiments for worlds W, V, resources $\vec{R}, \vec{Q}$, adversary A, simulator S, and transcript predicate ψ.

and due to the existing plethora of ROM-based results and the community's burgeoning focus on designing replacements for SHA-1 [141], the use of indifferentiability in the design and analysis of hash functions has become commonplace.

Despite this focus, the MRH framework is more broadly applicable. A few works have leveraged this, e.g.: to construct ideal ciphers from Feistel networks [61]; to define security of key-derivation functions in the multi-instance setting [28]; to unify various security goals for authenticated encryption [14]; or to formalize the goal of domain separation in the ROM [22]. Yet all of these applications of indifferentiability are about cryptographic primitives (i.e., objects that are non-interactive). To the best of our knowledge, ours is the first work to explicitly consider the application of indifferentiability to protocols.

Our conceptual starting point is a bit more general than MRH. In particular, we define indifferentiability in terms of the *world* in which the adversary finds itself, so named because of the common use of phrases like "real world", "ideal world", and "oracle worlds" when discussing security definitions. Formally, a world is a particular kind of *object* (defined in §2.1) that is constructed by connecting up a *game* [33] with a *scheme*, the former defining the security goal of the latter. The scheme is embedded within a *system* that specifies how the adversary and game interact with it, i.e., the scheme's execution environment.

Intuitively, when a world and an adversary are executed together, we can measure the probability of specific events occurring as a way to define adversarial success. Our MAIN$^{\psi}$ security experiment, illustrated in the left panel of Figure 1.1, captures this. The outcome of the experiment is 1 ("true") if the adversary A "wins", as determined by the output w of world W, and predicate ψ on the transcript tx of the adversary's queries also evaluates to 1. Along the lines of "penalty-style" definitions [133], the transcript predicate determines whether or not A's attack was valid, i.e., whether the attack constitutes a trivial win. (For example, if W captures IND-CCA security of an encryption scheme, then ψ would penalize decryption of challenge ciphertexts.)

Shared-Resource Indifferentiability and The Lifting Lemma. Also present in the experiment is a (possibly empty) tuple of *resources* $\vec{R}$, which may be called by both the world W and the adversary A. This captures embellishments to the base security experiment that may be used to prove security, but are not essential to the definition of security itself. An element of $\vec{R}$ might be an idealized object such as an RO [29], ideal cipher [61], or generic group [121]; it might be used to model global trusted setup, such as distribution of a common reference string [55]; or it might provide A (and W) with an oracle that solves some hard problem, such as the DDH oracle in the formulation of the Gap DH problem [109].

The result is a generalized notion of indifferentiability that we call *shared-resource indifferentiability.* The SR-INDIFF$^{\psi}$ experiment, illustrated in the right panel of Figure 1.1, considers an adversary's ability to distinguish some *real* world/resource pair $W/\vec{R}$ (read "W with $\vec{R}$") from a *reference* world/resource pair $V/\vec{Q}$ when the world and the adversary share access to the resources. The real world W exposes two interfaces to the adversary, denoted by subscripts W_1 and W_2, that we will call the *main* and *auxiliary* interfaces of W, respectively. The reference world V also exposes two interfaces (with the same monikers), although the adversary's access to the auxiliary interface of V is mediated by a *simulator* S. Likewise, the adversary has direct access to resources $\vec{R}$ in the real experiment, and S-mediated access to resources $\vec{Q}$ in the reference experiment.

The auxiliary interface captures what changes as a result of translating world $V/\vec{Q}$ into $W/\vec{R}$: the job of the simulator S is to "fool" the adversary into believing it is interacting with $W/\vec{R}$ when in fact it is interacting with $V/\vec{Q}$. Intuitively, if for a given adversary A there is a simulator S that successfully "fools" it, then this should yield a way to translate A's attack against $W/\vec{R}$ into an attack against $V/\vec{Q}$. This intuition is captured by our "lifting" lemma (Lemma 1, §2.3), which says that if $V/\vec{Q}$ is MAIN^{ψ}-secure and $W/\vec{R}$ is indifferentiable from $V/\vec{Q}$ (as captured by SR-INDIFF^{ψ}), then $W/\vec{R}$ is also MAIN^{ψ}-secure.

Games and The Preservation Lemma. For all applications in this dissertation, a world is specified in terms of two objects: the intended security goal of a scheme, formalized as a (single-stage [129]) game; and the system that specifies the execution environment for the scheme. In §2.4 we specify a world $W = \mathbf{Wo}(G, X)$ whose main interface allows the adversary to "play" the game G and whose auxiliary interface allows it to interact with the system X.

The world's auxiliary interface captures what "changes" from the reference experiment to the real one, and the main interface captures what stays the same. Intuitively, if a system X is indifferentiable from Y, then it ought to be the case that world $\mathbf{Wo}(G, X)$ is indifferentiable from $\mathbf{Wo}(G, Y)$, since in the former setting, the adversary might simply play the game G in its head. Thus, by Lemma 1, if Y is secure in the sense of G, then so is X. We formalize this intuition via a simple "preservation" lemma (Lemma 2, §2.4), which states that the indifferentiability of X from Y is "preserved" when access to X's (resp. Y's) main interface is mediated by a game G. As we show in §2.4, this yields the main result of MRH as a corollary (cf. [104, Theorem 1]).

Taken together, the lifting lemma and the preservation lemma allow us to prove some real system is secure by appealing directly to what is known a priori about the system from which it was derived. This significantly reduces the analytical effort of ensuring that translation does not result in an attack: instead of generating a fresh proof, it suffices to prove that that the real system is indifferentiable from the reference system. Another benefit of this approach is that it is (largely) agnostic to the security goal, since proving indifferentiability entails reasoning about the translation itself. As we will see, this provides a convenient way to vet changes to a scheme or its execution environment, without the need to descend into the particulars of the security goal.

Updated Pseudocode. An important feature of our framework is its highly expressive pseudocode. MRH define indifferentiability in terms of "interacting systems" formalized as sequences of conditional probability distributions (cf. [104, §3.1]). This abstraction, while extremely expressive, is much harder to work with than conventional cryptographic pseudocode. A contribution of this dissertation is to articulate an abstraction that provides much of the expressiveness of MRH, while preserving the level of rigor typical of game-playing proofs of security [33]. In §2.1 we formalize *objects*, which are used to define the various entities that run in security experiments, including games, adversaries, systems, and schemes.

Accounting for Unspecified Behavior. Finally, the translation framework also incorporates the PSP methodology of Rogaway and Stegers [130], allowing us to account for unspecified behavior in the objects we study. To do so, our experiments provide the world access the adversary's auxiliary interface (A_2, as shown in Figure 1.1), enabling the game and system to be defined so that the protocol's SD-oracle queries are answered by the adversary. We provide concrete demonstrations in Chapters 3 and 4.

1.3 Thesis Objectives and Organization

One of provable security's most important functions is to provide assurance for the translation of cryptography into practice, but it succeeds only to the extent that the system's formal specification faithfully captures its real-world behavior. Ensuring that it does bares a significant analytical cost, one that is often prohibitive to a fully rigorous treatment. Thus, the central thesis of this dissertation is as follows:

When evaluating the security impact of changes to a cryptographic scheme or its execution environment, it is possible to preserve the rigor of existing analysis without incurring the cost of a direct proof.

We put forward a theory for reconciling translation with existing results in which the scope of the new analysis is the translation itself. The objective of this dissertation is to demonstrate that narrowing the scope in this way substantially reduces the analytical effort required to strengthen the foundations of real-world cryptography.

Unspecified Behavior. We begin in Chapter 3 with an exploration of the limits of a priori security analysis. We give a formal treatment of the TLS 1.3 record layer [123], the component of TLS that is used to protect data sent during the course of the protocol. In our analysis (Theorem 2), we view the record layer as being only partially specified [130], allowing us to capture in its formal specification the full range of real-world behaviors that the standard admits. As a consequence, our result is much more robust than other studies with similar scope [114, 67]. Still, the TLS protocol is subject to changes that our treatment would not account for. By way of illustrating the practical value of the translation framework, we end this chapter with with a discussion of the limitations of our analysis.

Protocol Translation. We consider a translation to be "safe" if one can prove indifferentiability of the real system from the reference system. Chapter 4 presents our first real-world application of this idea, in which we consider the problem of translating a system by changing the scheme's specification. In particular, we specify a system **eCK** that formalizes the interaction of an adversary with an authenticated key-exchange (AKE) protocol in the extended Canetti-Krawczyk (eCK) model [99]. For real protocol $\mathit{\Pi}$ and reference protocol $\tilde{\mathit{\Pi}}$, indifferentiability of $\mathbf{eCK}(\mathit{\Pi})$ from $\mathbf{eCK}(\tilde{\mathit{\Pi}})$ lets us appeal to the security of $\tilde{\mathit{\Pi}}$ in order to argue that $\mathit{\Pi}$ is itself secure. We demonstrate this approach by designing a TLS extension that integrates a protocol for password-authenticated key-exchange (PAKE) [27] into the TLS handshake. We instantiate the extension with SPAKE2 [4] and exhibit a tight reduction from existing results for this PAKE [4, 20, 1] to its usage in TLS.

Environment Translation. Our second application, presented in Chapter 5, addresses the problem of translating the scheme's execution environment. In particular, we formalize a setting for key reuse in which multiple applications share access to a cryptographic API[4] that specifies the set of computations for one or more secret keys. In this setting, the reference system $\widetilde{\mathbf{API}}(\mathit{\Gamma})$ provides access to an API $\mathit{\Gamma}$ for the target application only, whereas the real system $\mathbf{API}(\mathit{\Gamma})$ provides access for multiple applications. The latter allows the adversary to mount an *exposed interface attack* against the target application by providing the adversary direct access to $\mathit{\Gamma}$, conservatively modeling cross protocol attacks against the target application that arise as a result of shared access to $\mathit{\Gamma}$. Using a property we call *context separability*, the lifting lemma and a variant of the preservation lemma (Lemma 4) allow us to argue that an application is secure under exposed interface attack by appealing to what has already been established by existing analysis. A special case of SR-INDIFF$^{\psi}$ security, context separability formalizes the property of $\mathit{\Gamma}$ that enables this reduction. This property evident in a variety of cryptographic standards, two of which we will study in detail: the RFC-standardized version of EdDSA [87]; and the Noise protocol framework [120].

Roadmap of the Remainder. The next section of this chapter discusses related work and describes various attacks that arise as a result of translation; the last section enumerates the preliminary work on

[4] Application Programming Interface.

which this dissertation is based. The translation framework is defined in Chapter 2; Chapters 3, 4, and 5 present the three studies described above; and we conclude in Chapter 6.

1.4 Related Work

Attacks Resulting From Translation. The class of attacks that result from translation is well-known, though not always as such. They are subtle, span multiple levels of abstraction, and crop up in a wide range of applications. Here we describe a few of the attacks that inspired our formal methodology.

Early versions of TLS have an optional feature known as "renegotiation" that allows the client and server to change the cryptographic parameters of an already established channel. If implemented, this feature leads to a man-in-the-middle attack in which the adversary forges a message from an authenticated client to a server. The first formal treatment of renegotiation is due to Giesen et al. [78], who describe this attack in the ACCE security model [85] and show that patching the protocol with the "renegotiation indication" extension [122] salvages security in this setting.

A cross protocol attack against TLS 1.2 was discovered by Mavrogiannopoulos et al. [105] that exploits an interaction between two key-exchange modes: classical DH [68], already in wide use at the time; and DH for groups over elliptic curves (ECDH), which were added to TLS later on [107]. TLS 1.2 allows the client and server to negotiate custom (EC)DH parameters as follows. In its ClientHello, the client indicates support for DH, ECDH, or both, and the ServerKeyExchange sent in response encodes the selected (EC)DH parameters, the server's (EC)DH key share, and a signature of the parameters and key share. Mavrogiannopoulos observe that the set of encodings of DH parameters/shares and the set of encodings of ECDH parameters/shares are non-disjoint, meaning it is possible for a client to misinterpret an ECDH key exchange as a DH key exchange. Under the right conditions, this ambiguity leads to a key-recovery attack that lets an active attacker impersonate the server to the client. While this attack is difficult to exploit, it does reveal a theoretical weakness in TLS 1.2. Indeed, provable security treatments of TLS 1.2 do not account for negotiation of custom EC(DH) parameters [45].

Keyless SSL [142] is a protocol deployed by Cloudflare that is used to proxy TLS connections between clients and servers, without the need to have the server's secret key on premise. In this protocol, the server exposes a signing API to mutually authenticated peers (i.e., trusted proxies) in order to allow them to terminate TLS on its behalf. To sign a handshake with a client, the proxy sends the message to be signed to the server, which responds with the signature. But instead of operating on the message msg itself, the API operates on a hash of the message $H(msg)$, which is an intermediate value in certain signature schemes used in TLS. Bhargavan et al. [40] show that this API could lead to a cross-protocol attack with QUIC [79], another widely-deployed secure channel protocol. In the absence of mutual authentication of the peer (which Keyless SSL provides), their attack would allow anyone with access to this API to impersonate a QUIC server.

There are several well-documented examples of API design flaws leading to insecure key reuse. Degabriele et al. [65] provide an analysis of the EMV[5] standard [72] for credit-card payments. To reduce overhead in this highly constrained environment, the API permits signing and decryption operations involving the same RSA secret key. The authors exhibit a practical forgery attack against the signature scheme that exploits oracle access to the decryption operation. An analysis by Künnemann et al. [97] points out a flaw in the API for Yubico's YubiHSM product that admits an oracle for a blockcipher keyed by the same key used to encrypt in CBC-mode. This leads to a fairly straightforward plaintext-recovery attack that substantially reduces the security of the HSM's[6] usage for key management. (The flaw has since been patched.)

[5] Europay, Mastercard, and Visa.

[6] Hardware Security Module.

In recent years, Intel and other chip manufacturers have moved to develop protocol standards for remote attestation of the state of a host, enabling a wide variety of trustworthy computing applications. Since the host is often an end-user system (e.g., a laptop or cellphone), ensuring that remote attestation preserves the user's privacy is paramount. There are a variety of direct anonymous attestation (DAA) schemes [50] designed for this purpose, and Intel's TPM[7] standard exposes an API designed to support many of them. Balancing flexibility and security in the API's design has proved to be challenging and has resulted in some subtle attack vectors [6, 52].

Sub-Protocol "Lifting". The lifting lemma (Lemma 1) owes its namesake to the formal treatment of downgrade resilience for key exchange of Bhargavan et al. [41]. Because widely deployed protocols like TLS allow communicants to support different feature sets, these protocols specify a mechanism by which the parties can agree on which set to use. Loosely, a key-exchange protocol is said to be downgrade resilient if no active attacker can force communicating parties to negotiate a feature set other than the one that would have been chosen in the absence of the attacker.

Recognizing the difficulty of precisely accounting for all of the details in complex standards, the authors devised the following analytical approach. They extract from the protocol's specifying document(s) the protocol core that is responsible for negotiating the feature set. They call this the "sub-protocol". Downgrade resilience is proved for the core protocol, then lifted to the full one by applying a composition theorem—the "downgrade security lifting" theorem [41, Theorem 2]—that transforms an attack against the full protocol into an attack against the core protocol. Intuitively, this theorem defines the set of full protocols whose downgrade security follows immediately from the downgrade security of the core protocol: part of the job of the analyst is to determine if the real protocol is in this set. (Incidentally, this step is akin to partitioning the standard into the PC and SD in the PSP framework [130].) As we show in §4.1.3, their theorem is essentially an information-theoretic analogue of our lifting lemma.

The Universal Composability (UC) Framework. MRH point out (cf. [104, §3.3]) that the notion of indifferentiability is inspired by ideas introduced in the UC framework [53]. There are conceptual similarities between UC (in particular, its generalization that allows for shared state [54]) and our framework, but the two are quite different in their details. We do not explore any formal relationship between frameworks, nor do we consider how one might modify UC to account for things that are naturally handled by ours, such as translation or unspecified behavior. However, we note that the two frameworks are quite different in their goals. The goal of the UC framework is to obtain simulation-style notions of security for cryptographic protocols that, via the universal composition theorem, modularize their analysis with respect to concrete attacks. In contrast, our primary objective is to reason about secure translation.

1.5 Preliminary Work

This thesis is based on three papers by the author: the first, referred to as PS18, is the basis of Chapter 3; the second, PS19, is the basis of Chapter 5; and the third, PS20, describes the translation framework of Chapter 2 and its application to protocol translation in Chapter 4.

PS18 [116] "Partially specified channels: The TLS 1.3 record layer without elision." CCS 2018.

PS19 [117] "Security in the presence of key reuse: Context-separable interfaces and their applications." CRYPTO 2019.

PS20 [118] "Quantifying the security cost of migrating protocols to practice." CRYPTO 2020.

[7] Trusted Platform Module.

There are substantial differences between the original publications of PS18 [116] and PS19 [117] and their presentation here. First, there is a bug in the original analysis of the record layer in PS18, which we resolve by placing mild restrictions on the behavior of the SD oracle (cf. Remark 4). Second, we reformulate the definitions of PS19 in the more general setting of SR-INDIFF$^{\psi}$ security, which yields a far simpler and more general treatment of exposed interface attacks (cf. Remark 7). Along the way, we fix an error in the derivation of the bound for [117, Theorem 1].

Chapter 2

The Translation Framework

This chapter describes the formal foundation of this dissertation. We begin in §2.1 by defining objects, our abstraction of the various entities run in a security experiment; in §2.2 we define our base experiment and formalize shared-resource indifferentiability; in §2.3 we state and prove the lifting lemma, the central technical tool of this work; and in §2.4 we formalize the class of security goals to which our framework applies.

Notation. When X is a random variable we let $\Pr\left[\, X = v \,\right]$ denote the probability that X is equal to v; we write $\Pr\left[\, X \,\right]$ as shorthand for $\Pr\left[\, X = 1 \,\right]$. We let $x \leftarrow y$ denote assignment of the value of y to variable x. When $\mathcal{X}$ is a finite set we let $x \twoheadleftarrow \mathcal{X}$ denote random assignment of an element of $\mathcal{X}$ to x according to the uniform distribution.

A *string* is an element of $\{0,1\}^*$; a *tuple* is a finite sequence of symbols separated by commas and delimited by parentheses. Let ε denote the empty string, $(\,)$ the empty tuple, and $(z,)$ the singleton tuple containing z. We sometimes (but not always) denote a tuple with an arrow above the variable (e.g., $\vec{x}$). Let $|x|$ denote the length of a string (resp. tuple) x. Let x_i and $x[i]$ denote the i-th element of x. Let $x \,\|\, y$ denote concatenation of x with string (resp. tuple) y. When x is a string, let $x[i{:}j]$ denote the sub-string $x_i \,\|\, \cdots \,\|\, x_j$ of x. If $i \notin [1..j]$ or $j \notin [i..|x|]$, then define $x[i{:}j] = \bot$. Let $x[i{:}] = x[i{:}|x|]$ and $x[{:}j] = x[1{:}j]$. We write $x \preceq y$ if string x is a prefix of string y, i.e., there exists some r such that $x \,\|\, r = y$. Let $y \,\%\, x$ denote the "remainder" r after removing the prefix x from y; if $x \not\preceq y$, then define $y \,\%\, x = \varepsilon$ (cf. [48]). When x is a tuple we let $x \,.\, z = (x_1, \ldots, x_{|x|}, z)$ so that z is "appended" to x. We write $z \in x$ if $(\exists\, i)\, x_i = z$. Let $[i..j]$ denote the set of integers $\{i, \ldots, j\}$; if $j < i$, then define $[i..j]$ as $\emptyset$. Let $[n] = [1..n]$.

For all sets $\mathcal{X}$ and functions $f, g : \mathcal{X} \to \{0,1\}$, define function $f \wedge g$ as the map $[f \wedge g](x) \mapsto f(x) \wedge g(x)$ for all $x \in \mathcal{X}$. We denote a group as a pair $(\mathcal{G}, *)$, where $\mathcal{G}$ is the set of group elements and $*$ denotes the group action. Logarithms are base-2 unless otherwise specified.

2.1 Objects

Our goal is to preserve the expressiveness of the MRH framework [104] while providing the level of rigor of code-based game-playing arguments [33]. To strike this balance, we will need to add a bit of machinery to standard cryptographic pseudocode. Objects provide this.

Each object has a *specification* that defines how it is used and how it interacts with other objects. We first define specifications, then describe how to *call* an object in an experiment and how to *instantiate* an object. Pseudocode in this dissertation will be typed (along the lines of Rogaway and Stegers [130]), so we enumerate the available types in this section. We finish by defining various properties of objects that will be used in the remainder.

```
astub       →  _ | stub
astubs      →  astub | astub, astubs
avar        →  _ | var
avars       →  avar | avar, avars
dec         →  var typedvars
decs        →  ε | dec | dec; decs
interface   →  interface type: {ops}
op          →  op^oracles (pattern) otype: {block}
ops         →  ε | op | op; ops
otype       →  ε | type
oracles     →  ε | astubs
pattern     →  ε | ... | literal | avars type | (pattern) | patterns
patterns    →  pattern | pattern, patterns
proc        →  procedure type(vars): {block}
spec        →  spec type: {decs; ops}
typedvars   →  vars type | vars type, typedvars
vars        →  var | var, vars
```

```
spec Ro:
1  var $\mathcal{X}, \mathcal{Y}$ set, $q, p$ int
2  var $T$ table, $i, j$ int
3  op (SETUP): $T \leftarrow [\,]$; $i, j \leftarrow 0$
4  op ($x$ elem$_{\mathcal{X}}$):
5    if $i \geq q$ then ret $\perp$
6    if $T[x] = \perp$ then
7      $i \leftarrow i + 1$; $T[x] \twoheadleftarrow \mathcal{Y}$
8    ret $T[x]$
9  op (SET, $M$ object):
10   var $x$ elem$_{\mathcal{X}}$, $y$ elem$_{\mathcal{Y}}$
11   if $j \geq p$ then ret $\perp$
12   $j \leftarrow j + 1$; $((x, y), \sigma) \leftarrow M(\,)$
13   $T[x] \leftarrow y$
14   ret $((x, y), \sigma)$
```

Figure 2.1: Left: Context-free grammar for specifications. Production begins with term spec. Variables type, var, literal, block, and stub are undefined. Code blocks will usually be denoted by indentation rather than "{" and "}". The semicolon ";" will usually be denoted by a new line. Right: Specification of a random oracle (RO) object. When instantiated, variables $\mathcal{X}$ and $\mathcal{Y}$ determine the domain and range of the RO, and integers q and p determine, respectively, the maximum number of distinct RO queries, and the maximum number of RO-programming queries (via the SET-operator), (cf. Def. 7).

2.1.1 Specifications

The relationship between a specification and an object is analogous to (but far simpler than) the relationship between a class and a class instance in object-oriented programming languages like Python or C++. A specification defines an ordered sequence of *variables* stored by an object—these are akin to attributes in Python—and an ordered sequence of *operators* that may be called by other objects—these are akin to methods. We refer to the sequence of variables as the object's *state* and to the sequence of operators as the object's *interface*.

We provide an example of a specification in Figure 2.1. Spec **Ro** is used throughout this work to model functions as random oracles (ROs) [29]. It declares seven variables, $\mathcal{X}$, $\mathcal{Y}$, q, p, T, i, and j, as shown on lines 1-2 in Figure 2.1. (We will use shorthand for line references in the remainder, e.g., "2.1:1-2" rather than "lines 1-2 in Figure 2.1".) Each variable has an associated type: $\mathcal{X}$ and $\mathcal{Y}$ have type **set**, q, p, i, and j have type **int**, and T has type **table**. Variable declarations are denoted by the keyword "var", while operator definitions are denoted by the keyword "op". Spec **Ro** defines three operators: the first, the SETUP-operator (2.1:3), initializes the RO's state; the second operator (2.1:4-8) responds to standard RO queries; and the third, the SET-operator (2.1:9-14), is used to "program" the RO [77].

Pseudocode. The syntax of specifications is given by the context-free grammar in Figure 2.1. We have omitted the production rules for type, var, literal, and block since they are standard. Briefly, a var denotes a variable. A type denotes a type, e.g., **set**, **int**, **table**, or **elem**$_{\mathcal{X}}$, or the name of a specification, e.g., **Ro**. A literal is a finite sequence of symbols from some alphabet, e.g., an integer or a bit string. We often write bit-string literals as alphanumeric strings, e.g., SETUP or SET, which are understood to be distinct elements of $\{0, 1\}^*$. A block is a sequence of statements such as variable declarations, (random) assignment statements, if-then-else blocks, for-loops, return statements, etc. A stub is an *interface oracle*, written calligraphically like $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, and so on. (Note that we also write sets calligraphically.) It is the name used by the operator to refer to an oracle passed to it, as we describe below.

2.1.2 Calling an Object

An object is *called* by providing it with oracles and passing arguments to it. An oracle is always an interface, i.e., a sequence of operators defined by an object. The statement "$out \leftarrow obj^{\mathbf{I}_1, \ldots, \mathbf{I}_m}(in_1, \ldots, in_n)$" means to invoke one of obj's operators on input of $in_1, \ldots, in_n$ and with oracle access to interfaces $\mathbf{I}_1, \ldots, \mathbf{I}_m$ and set variable out to the value returned by the operator. Objects will usually have many operators, so we must specify the manner in which the responding operator is chosen. For this purpose we will adopt a convention inspired by "pattern matching" in functional languages like Haskell and Rust. Syntactically, a pattern is defined by the $\textsf{pattern}$ term in Figure 2.1. It is comprised of a tuple of literals, typed variables, and nested tuples. A value is said to *match* a pattern if they have the same type and the literals are equal. For example, value val matches pattern $(\_\,\mathbf{elem}_{\mathcal{X}})$ if val has type $\mathbf{elem}_{\mathcal{X}}$. (The symbol "$\_$" contained in the pattern denotes an anonymous variable.) Hence, if object R is specified by **Ro** and x has type $\mathbf{elem}_{\mathcal{X}}$, then the expression "$R(x)$" calls R's second operator (2.1:4-8). We write "$val \sim pat$" if the value of variable val matches pattern pat.

Calls to objects are evaluated as follows. In the order in which they are defined, check each operator of the object's specification if the input matches the operator's pattern. If so, then execute the operator until a return statement is reached and assign the return value to the output. If no return statement is reached, or if val does not match an operator, then return $\bot$.

Let us consider an illustrative example. Let Π be an object that implements Schnorr's signature scheme [135] for a group $(\mathcal{G}, \cdot)$ as specified in Figure 2.2. The expression $\Pi(\textsf{GEN})$ calls Π's first operator, which generates a fresh key pair. If $s \in \mathbb{Z}$ and $msg \in \{0,1\}^*$, then expression $\Pi_s^H(\textsf{SIGN}, msg)$ evaluates the third operator, which computes a signature (x,t) of message msg under secret key s (we will often write the first argument as a subscript). The call to interface oracle $\mathcal{H}$ on line 2.2:5 is answered by object H. (Presumably, H is a hash function with domain $\mathcal{G} \times \{0,1\}^*$ and range $\mathbb{Z}_{|\mathcal{G}|}$.) If $PK \in \mathcal{G}$, $msg \in \{0,1\}^*$, and $x, t \in \mathbb{Z}$, then expression $\Pi_{PK}^H(\textsf{VERIFY}, msg, (x,t))$ evaluates the second operator. On an input that does not match any of these patterns—in particular, one of $(\textsf{GEN})$, $(\_\,\mathbf{elem}_{\mathcal{G}}, \textsf{VERIFY}, \_\,\mathbf{str}, (\_, \_\,\mathbf{int}))$, or $(\_\mathbf{int}, \textsf{SIGN}, \_\,\mathbf{str})$—the object returns $\bot$. For example, $\Pi^{\mathbf{I}_1,\ldots,\mathbf{I}_m}(\textsf{foo bar}) = \bot$ for any $\mathbf{I}_1, \ldots, \mathbf{I}_m$.

It is up to the caller to ensure that the correct number of interfaces is passed to the operator. If the number of interfaces passed is less than the number of oracles named by the operator, then calls to the remaining oracles are always answered with $\bot$; if the number of interfaces is more than the number of oracles named by the operator, then the remaining interfaces are simply ignored by the operator.

Explanation. We will see examples of pattern matching in action throughout this dissertation. For now, the important takeaway is that calling an object results in one (or none) of its operators being invoked: which one is invoked depends on the type of input and the order in which the operators are defined.

Because these calling conventions are more sophisticated than usual, let us take a moment to explain their purpose. Theorem statements in this work will often quantify over large sets of objects whose functionality is unspecified. These conventions ensure that doing so is always well-defined, since any object can be called on any input, regardless of the input type. We could have dealt with this differently: for example, in their adaptation of indifferentiability to multi-staged games, Ristenpart et al. require a similar convention for functionalities and games (cf. "unspecified procedure" in [129, §2]). Our hope is that the higher level of rigor of our formalism will ease the task of verifying proofs of security in our framework.

2.1.3 Instantiating an Object

An object is instantiated by passing arguments to its specification. The statement "$obj \leftarrow \mathbf{Object}(in_1, \ldots, in_m)$" means to create a new object obj of type **Object** and initialize its state by setting $obj.var_1 \leftarrow in_1$, ..., $obj.var_m \leftarrow in_m$, where $var_1, \ldots, var_m$ are the first m variables declared by **Object**. If the number of arguments passed is less than the number of variables declared, then the remaining variables are uninitialized. For example, the statement "$R \leftarrow \mathbf{Ro}(\mathcal{X}, \mathcal{Y}, q, p, [\,], 0, 0)$" initializes R by setting $R.\mathcal{X} \leftarrow \mathcal{X}$, $R.\mathcal{Y} \leftarrow \mathcal{Y}$, $R.q \leftarrow q$, $R.p \leftarrow p$, $R.T \leftarrow [\,]$, $R.i \leftarrow 0$, and $R.j \leftarrow 0$. The statement "$R \leftarrow \mathbf{Ro}(\mathcal{X}, \mathcal{Y}, q, p)$" sets $R.\mathcal{X} \leftarrow \mathcal{X}$, $\mathcal{R}.\mathcal{Y} \leftarrow \mathcal{Y}$, $R.q \leftarrow q$, and $R.p \leftarrow p$, but leaves T, i, and j uninitialized. Object can also be copied: the statement "$new \leftarrow obj$" means to instantiate a new object new with

```
spec Schnorr:
1  op (GEN): $s \twoheadleftarrow \mathbb{Z}_{|\mathcal{G}|}$; $PK \leftarrow g^s$; ret $(PK, s)$
2  op$^{\mathcal{H}}$ ($PK$ elem$_{\mathcal{G}}$, VERIFY, $msg$ str, $(x, t$ int$)$):
3      ret $t \equiv \mathcal{H}(g^x \cdot PK^t, msg) \pmod{|\mathcal{G}|}$
4  op$^{\mathcal{H}}$ ($s$ int, SIGN, $msg$ str):
5      $r \twoheadleftarrow \mathbb{Z}_{|\mathcal{G}|}$; $t \leftarrow \mathcal{H}(g^r, msg)$
6      ret $(r - st, t)$
```

Figure 2.2: Specification of Schnorr's signature scheme.

specification **Object** and set $new.var_1 \leftarrow obj.var_1, \ldots, new.var_n \leftarrow obj.var_n$, where $var_1, \ldots, var_n$ is the sequence of variables declared by obj's specification.

2.1.4 Types

We now enumerate the types available in our pseudocode. An object has type **object**. A set of values of type **any** (defined below) has type **set**; we let $\emptyset$ denote the empty set. A variable of type **table** stores a table of key/value pairs, where keys and values both have type **any**. If T is a table, then we let T_k and $T[k]$ denote the value associated with key k in T; if no such value exists, then $T_k = \bot$. We let $[\,]$ denote the empty table.

When the value of a variable x is an element of a computable set $\mathcal{X}$, we say that x has type $\mathbf{elem}_{\mathcal{X}}$. We define type **int** as an alias of $\mathbf{elem}_{\mathbb{Z}}$, type **bool** as an alias of $\mathbf{elem}_{\{0,1\}}$, and type **str** as an alias of $\mathbf{elem}_{\{0,1\}^*}$. We define type **any** recursively as follows. A variable x is said to have type **any** if: it is equal to $\bot$ or $(\,)$; has type **set**, **table**, or $\mathbf{elem}_{\mathcal{X}}$ for some computable set $\mathcal{X}$; or it is a tuple of values of type **any**.

Specifications declare the type of each variable of an object's state. The types of variables that are local to the scope of an operator need not be explicitly declared, but their type must be inferable from their initialization (that is, the first use of the variable in an assignment statement). If a variable is assigned a value of a type other than the variable's type, then the variable is assigned $\bot$. Variables that are declared but not yet initialized have the value $\bot$. For all $\mathbf{I}_1, \ldots, \mathbf{I}_m, in_1, \ldots, in_n$ the expression "$\bot^{\mathbf{I}_1,\ldots,\mathbf{I}_m}(in_1, \ldots, in_n)$" evaluates to $\bot$. We say that $x = \bot$ or $\bot = x$ if variable x was previously assigned $\bot$. For all other expressions, our convention will be that whenever $\bot$ is an input, the expression evaluates to $\bot$.

2.1.5 Properties of Operators and Objects

An operator is called *deterministic* if its definition does not contain a random assignment statement; it is called *stateless* if its definition contains no assignment statement in which one of the object's variables appears on the left-hand side; and an operator is called *functional* if it is deterministic and stateless. Likewise, an object is called deterministic (resp. stateless or functional) if each operator, with the exception of the SETUP-operator, is deterministic (resp. stateless or functional). (We make an exception for the SETUP-operator in order to allow trusted setup of objects executed in our experiments. See §2.2 for details.)

Resources. Let $t \in \mathbb{N}$. An operator is called *t-time* if it always halts in t time steps regardless of its random choices or the responses to its queries; we say that an operator is *halting* if it is t-time for some $t < \infty$. Our convention will be that an operator's runtime includes the time required to evaluate its oracle queries. Let $\vec{q} \in \mathbb{N}^*$. An operator is called *$\vec{q}$-query* if it makes at most $\vec{q}_1$ calls to its first oracle, $\vec{q}_2$ to its second, and so on. We extend these definitions to objects, and say that an object is t-time (resp. halting or $\vec{q}$-query) if each operator of its interface is t-time (resp. halting or $\vec{q}$-query).

Exported Operators. An operator f_1 is said to *shadow* operator f_2 if: (1) f_1 appears first in the sequence of operators defined by the specification; and (2) there is some input that matches both f_1 and f_2. For example, an operator with pattern (x **any**) would shadow an operator with pattern (y **str**), since y is of type **str** and **any**. An object is said to export a *pat-type*-operator if its specification defines a non-shadowed operator that, when run on an input matching pattern *pat*, always returns a value of type *type*. Let $\mathcal{X}$ and $\mathcal{Y}$ be computable sets. An object F *computes* a function $f : \mathcal{X} \to \mathcal{Y}$ if F is halting, functional, exports an $(\_\ \mathbf{elem}_{\mathcal{X}})$-$\mathbf{elem}_{\mathcal{Y}}$-operator, and $F(x) = f(x)$ holds for every $x \in \mathcal{X}$.

2.2 Experiments and Indifferentiability

This section describes our core security experiments. An experiment connects up a set of objects in a particular way, giving each object oracle access to interfaces (i.e., sequences of operators) exported by other objects. An object's *i-interface* is the sequence of operators whose patterns are prefixed by literal i. We sometimes write i as a subscript, e.g., "$X_i(\cdots)$" instead of "$X(i, \cdots)$" or "$X(i, (\cdots))$". We refer to an object's 1-interface as its *main* interface and to its 2-interface as its *auxiliary* interface.

```
procedure $\mathbf{Real}^{\Phi}_{W/\vec{R}}(A)$:
 1  $A(\textsf{SETUP})$; $W(\textsf{SETUP})$
 2  for $i \gets 1$ to $u$ do $R_i(\textsf{SETUP})$
 3  $tx \gets (\,)$; $a \gets A_1^{\mathbf{W}_1,\mathbf{W}_2,\mathbf{R}}(\textsf{OUT})$
 4  $w \gets W_1(\textsf{WIN})$; ret $\Phi(tx, a, w)$

procedure $\mathbf{W}(i, x)$:
 5  $y \gets W_i^{\mathbf{A}_2,\mathbf{R}}(x)$; $tx \gets tx \,.\, (i, x, y)$; ret $y$

procedure $\mathbf{A}(i, x)$:
 6  if $S = \bot$ then ret $A_i^{\mathbf{W}_1,\mathbf{W}_2,\mathbf{R}}(x)$  // Real
 7  ret $A_i^{\mathbf{W}_1,\mathbf{S}_2,\mathbf{S}_3}(x)$  // Ref

procedure $\mathbf{Ref}^{\Phi}_{W/\vec{R}}(A, S)$:
 8  $S(\textsf{SETUP})$; $A(\textsf{SETUP})$; $W(\textsf{SETUP})$
 9  for $i \gets 1$ to $u$ do $R_i(\textsf{SETUP})$
10  $tx \gets (\,)$; $a \gets A_1^{\mathbf{W}_1,\mathbf{S}_2,\mathbf{S}_3}(\textsf{OUT})$
11  $w \gets W_1(\textsf{WIN})$; ret $\Phi(tx, a, w)$

procedure $\mathbf{R}(i, x)$:
12  ret $R_i(x)$

procedure $\mathbf{S}(i, x)$:
13  ret $S_i^{\mathbf{W}_2,\mathbf{R}}(x)$
```

Figure 2.3: Real and reference experiments for world W, resources $\vec{R} = (R_1, \ldots, R_u)$, adversary A, and simulator S.

A *resource* is a halting object. A *simulator* is a halting object. An *adversary* is a halting object that exports a $(1, \textsf{OUT})$-**bool**-operator, which means that on input of $(\textsf{OUT})$ to its main interface, it outputs a bit. This operator is used to in order to initiate the adversary's attack. The attack is formalized by the adversary's interaction with another object, called the *world*, which codifies the system under attack and the adversary's goal. Formally, a world is a halting object that exports a functional $(1, \textsf{WIN})$-**bool**-operator, which means that on input of $(\textsf{WIN})$ to its main interface, the world outputs a bit that determines if the adversary has won. The operator being functional means this decision is made deterministically and in a "read-only" manner, so that the object's state is not altered. (These features are necessary to prove the lifting lemma in §2.3.)

2.2.1 MAIN Security

Security experiments are formalized by the execution of procedure **Real** defined in Figure 2.3 for adversary A in world W with shared resources $\vec{R} = (R_1, \ldots, R_u)$. In addition, the procedure is parameterized by a function Φ. The experiment begins by "setting up" each object by running $A(\textsf{SETUP})$, $W(\textsf{SETUP})$, and $R_i(\textsf{SETUP})$ for each $i \in [u]$. This allows for trusted setup of each object before the attack begins. Next, the procedure runs A with oracle access to procedures $\mathbf{W}_1$, $\mathbf{W}_2$, and $\mathbf{R}$, which provide A with access to, respectively, W's main interface, W's auxiliary interface, and the resources $\vec{R}$.

Figure 1.1 illustrates which objects have access to which interfaces. The world W and adversary A share access to the resources $\vec{R}$. In addition, the world has access to the auxiliary interface of A (2.3:5), which allows us to formalize security properties in the PSP setting [130]. (Interestingly, it also turns out to be essential to MRH's argument of the necessity of indifferentiability; see Proposition 2.) Each query to $\mathbf{W}_1$ or $\mathbf{W}_2$ by A is recorded in a tuple tx called the *experiment transcript* (2.3:5). The outcome of the experiment is $\Phi(tx, a, w)$, where a is the bit output by A and w is the bit output by W. The MAIN$^{\psi}$ security notion, defined below, captures an adversary's advantage in "winning" in a given world, where what it means to "win" is defined by the world itself. The validity of the attack is defined by a function ψ, called the *transcript predicate*: in the MAIN$^{\psi}$ experiment, we define Φ so that $\mathbf{Real}^{\Phi}_{W/\vec{R}}(A) = 1$ holds if A wins and $\psi(tx) = 1$ holds.

Definition 1 (MAIN$^{\psi}$ security)**.** Let W be a world, $\vec{R}$ be resources, and A be an adversary. Let ψ be a transcript predicate, and let $\mathsf{win}^{\psi}(tx, a, w) := (\psi(tx) = 1) \wedge (w = 1)$. The MAIN$^{\psi}$ advantage of A in attacking $W/\vec{R}$ is

$$\mathbf{Adv}^{\mathrm{main}^{\psi}}_{W/\vec{R}}(A) := \Pr\big[\, \mathbf{Real}^{\mathsf{win}^{\psi}}_{W/\vec{R}}(A) \,\big] \,.$$

Informally, we say that $W/\vec{R}$ is *ψ-secure* if the MAIN$^{\psi}$ advantage of every efficient adversary is small. Note that advantage for indistinguishability-style security notions is defined by normalizing MAIN$^{\psi}$ advantage (e.g., Def. 10 or Def. 17). □

This measure of advantage is only meaningful if ψ is efficiently computable, since otherwise a computationally bounded adversary may lack the resources needed to determine if its attack is valid. Following Rogaway-Zhang (cf. computability of "fixedness" in [133, §2]) we will require $\psi(tx)$ to be efficiently computable given the entire transcript, except the response to the last query. Intuitively, this exception ensures that, at any given moment, the adversary "knows" whether its next query is valid before making it.

Definition 2 (Transcript-predicate computability)**.** Let ψ be a transcript predicate. Object F *computes* ψ if it is halting, functional, and $F(\bar{tx}) = \psi(tx)$ holds for all transcripts tx, where $\bar{tx} = (tx_1, \ldots, tx_{q-1}, (i_q, x_q, \perp))$, $q = |tx|$, and $(i_q, x_q, \_) = tx_q$. We say that ψ is *computable* if there is an object that computes it. We say that ψ is t-time computable if there is a t-time object F that computes it. Informally, we say that ψ is efficiently computable if it is t-time computable for small t. □

Shorthand. In the remainder we write "$W/\vec{R}$" as "W/H" when "$\vec{R} = (H,)$", i.e., when the resource tuple is a singleton containing H. Similarly, we write "$W/\vec{R}$" as "W" when $\vec{R} = (\,)$, i.e., when no shared resources are available. We write "win" instead of "win^{ψ}" whenever ψ is defined so that $\psi(tx) = 1$ for all transcripts tx. Correspondingly, we write "MAIN" for the security notion obtained by letting $\Phi = \mathsf{win}$.

2.2.2 SR-INDIFF Security

The **Real** procedure executes an adversary in a world that shares resources with the adversary. We are interested in the adversary's ability to distinguish this "real" experiment from a "reference" experiment in which we change the world and/or resources with which the adversary interacts. To that end, Figure 2.3 also defines the **Ref** procedure, which executes an adversary in a fashion similar to **Real** except that a simulator S mediates the adversary's access to the resources and the world's auxiliary interface. In particular, A's oracles $\mathbf{W}_2$ and $\mathbf{R}$ are replaced with $\mathbf{S}_2$ and $\mathbf{S}_3$ respectively (2.3:7 and 10), which run S with access to $\mathbf{W}_2$ and $\mathbf{R}$ (2.3:13). SR-INDIFF$^{\psi}$ advantage, defined below, measures the adversary's ability to distinguish between a world $W/\vec{R}$ in the real experiment and another world $V/\vec{Q}$ in the reference experiment.

Definition 3 (SR-INDIFF$^{\psi}$ security)**.** Let W, V be worlds, $\vec{R}, \vec{Q}$ be resources, A be an adversary, and S be a simulator. Let ψ be a transcript predicate and let $\mathsf{out}^{\psi}(tx, a, w) := (\psi(tx) = 1) \wedge (a = 1)$. Define the SR-INDIFF$^{\psi}$ advantage of adversary A in differentiating $W/\vec{R}$ from $V/\vec{Q}$ relative to S as

$$\mathbf{Adv}^{\mathrm{sr\text{-}indiff}^{\psi}}_{W/\vec{R}, V/\vec{Q}}(A, S) := \Pr\left[\, \mathbf{Real}^{\mathsf{out}^{\psi}}_{W/\vec{R}}(A) \,\right] - \Pr\left[\, \mathbf{Ref}^{\mathsf{out}^{\psi}}_{V/\vec{Q}}(A, S) \,\right] .$$

By convention, the runtime of A is the runtime of $\mathbf{Real}^{\mathsf{out}^{\psi}}_{W/\vec{R}}(A)$. Informally, we say that $W/\vec{R}$ is *ψ-indifferentiable* from $V/\vec{Q}$ if for every efficient A there exists an efficient S for which the SR-INDIFF$^{\psi}$ advantage of A is small. □

Shorthand. We write "out" instead of "out^{ψ}" when ψ is defined so that $\psi(tx) = 1$ for all tx. Correspondingly, we write "SR-INDIFF" for the security notion obtained by letting $\Phi = \mathsf{out}$.

2.2.3 Non-Degenerate Adversaries

When defining security, it is typical to design the experiment so that it is guaranteed to halt. Indeed, there are pathological conditions under which $\mathbf{Real}^{\Phi}_{W/\vec{R}}(A)$ and $\mathbf{Ref}^{\Phi}_{W/\vec{R}}(A, S)$ do not halt, even if each of the constituent objects is halting (as defined in §2.1.5). This is because infinite loops are possible: in response to a query from adversary A, the world W is allowed to query the adversary's auxiliary interface A_2; the responding operator may call W in turn, which may call A_2, and so on. Consequently, the event that $\mathbf{Real}^{\Phi}_{W/\vec{R}}(A) = 1$ (resp. $\mathbf{Ref}^{\Phi}_{W/\vec{R}}(A, S) = 1$) must be regarded as the event that the real (resp. reference) experiment halts and outputs 1. Defining advantage this way creates obstacles for quantifying resources of a security reduction, so it will be useful to rule out infinite loops.

We define the class of *non-degenerate (n.d.) adversaries* as those that respond to main-interface queries using all three oracles—the world's main interface, the world's aux.-interface, and the resources—but respond to aux.-interface queries using only the resource oracle. To formalize this behavior, we define n.d. adversaries in terms of an object that is called in response to main-interface queries, and another object that is called in response to aux.-interface queries.

```
spec $\mathbf{NoDeg}$:  // $\mathcal{W}$ points to $W_1$; $\mathcal{W}'$ to $W_2$;
                        // $\mathcal{R}$ to resources
    var $M, SD$ object
    op (SETUP): $M$(SETUP); $SD$(SETUP)
    op$^{\mathcal{W},\mathcal{W}',\mathcal{R}}$ $(1, x$ any$)$: ret $M^{\mathcal{W},\mathcal{W}',\mathcal{R}}(x)$
    op$^{\mathcal{W},\mathcal{W}',\mathcal{R}}$ $(2, x$ any$)$: ret $SD^{\mathcal{R}}(x)$
```

```
spec $\mathbf{Sh}_\lambda$:  // $\mathcal{A}$ points to $A_2$ in the experiment.
    var $W, R_1, \ldots, R_\lambda$ object
    op (SETUP): $W$(SETUP)
        for $i \leftarrow 1$ to $\lambda$ do $R_i$(SETUP)
    op$^{\mathcal{A}}$ $(1,$ WO$, x$ any$)$: ret $W_1^{\mathbf{R},\mathcal{A}}(x)$
    op$^{\mathcal{A}}$ $(2,$ WO$, x$ any$)$: ret $W_2^{\mathbf{R},\mathcal{A}}(x)$
    op$^{\mathcal{A}}$ $(2,$ RO$, i$ int$, x$ any$)$: ret $R_i(x)$
procedure $\mathbf{R}(i, x)$:
    ret $R_i(x)$
```

Figure 2.4: Left: Specification of n.d. (non-degenerate) adversaries. Right: Specification $\mathbf{Sh}_\lambda$ used in Proposition 1.

Definition 4 (Non-degenerate adversaries)**.** An adversary A is called *non-degenerate (n.d.)* if there exist a halting object M that exports an (OUT)-**bool**-operator and a halting, functional object SD for which $A = \mathbf{NoDeg}(M, SD)$ as specified in Figure 2.4. We refer to M as the *main algorithm* and to SD as the *auxiliary algorithm.* □

Observe that we have also restricted n.d. adversaries so that the main and auxiliary algorithms do not share state; and we have required that the auxiliary algorithm is functional (i.e., deterministic and stateless). These measures are not necessary, strictly speaking, but they will be useful for security proofs. Their purpose is primarily technical, as they do not appear to be restrictive in a practical sense. (They do not limit any of the applications considered in this thesis. Incidentally, we note that Rogaway and Stegers make similar restrictions in [130, §5]; see Remark 4.)

2.2.4 Equivalence of SR-INDIFF and INDIFF

An analogue of MRH's notion of indifferentiability is obtained by removing the shared resources from the SR-INDIFF experiment, i.e., letting $\vec{R}, \vec{Q} = (\,)$.

Definition 5 (INDIFF$^\psi$ security)**.** Let W, V be worlds, A an adversary, S a simulator, and ψ a transcript predicate. Let $\mathbf{Adv}^{\text{indiff}^\psi}_{W,V}(A, S) := \mathbf{Adv}^{\text{sr-indiff}^\psi}_{W,V}(A, S)$ denote the INDIFF$^\psi$ advantage of A in differentiating W from V relative to S. □

In this sense, SR-INDIFF$^\psi$ security can be viewed as a generalization of the standard notion. An alternative view is that shared-resource indifferentiability merely captures a particular class of indifferentiability problems. Indeed, for world W and resources $\vec{R} = (R_1, \ldots, R_u)$, Figure 2.4 specifies a world $\hat{W} = \mathbf{Sh}_u(W, R_1, \ldots, R_u)$ that is functionally equivalent to $W/\vec{R}$, except that the resources are codified by the world $\hat{W}$ rather than defined externally.

Proposition 1. *Let ψ be a transcript predicate, $f : \mathbb{N} \to \mathbb{N}$ be a function, W, V be worlds, and $\vec{R} = (R_1, \ldots, R_u), \vec{Q} = (Q_1, \ldots, Q_v)$ be resources. Let $\hat{W} = \mathbf{Sh}_u(W, R_1, \ldots, R_u)$ and $\hat{V} = \mathbf{Sh}_v(V, Q_1, \ldots, Q_v)$. Let A be a t_A-time, n.d. adversary, let T be a t_T-time simulator, and suppose that ψ is $f(t_A)$-time computable.*

(1) *There exist a $O(t_A)$-time, n.d. adversary B, $O(t_T)$-time simulator S, and $[O(t_A) + f(t_A)]$-time computable transcript-predicate ϕ such that $\mathbf{Adv}^{\text{indiff}^\psi}_{\hat{W},\hat{V}}(A, S) \le \mathbf{Adv}^{\text{sr-indiff}^\phi}_{W/\vec{R}, V/\vec{Q}}(B, T)$.*

(2) *There exist a $O(t_A)$-time, n.d. adversary B, $O(t_T)$-time simulator S, and $[O(t_A) + f(t_A)]$-time computable transcript-predicate ϕ such that $\mathbf{Adv}^{\text{sr-indiff}^\psi}_{W/\vec{R}, V/\vec{Q}}(A, S) \le \mathbf{Adv}^{\text{indiff}^\phi}_{\hat{W},\hat{V}}(B, T)$.*

Proof. We begin with claim (1). Adversary B is specified as follows. On input of (SETUP), run A(SETUP). On input of $(1,$ OUT$)$ with interface oracles $\mathcal{W}, \mathcal{W}', \mathcal{R}$ corresponding to the world's main interface, the world's aux. interface, and the resource interface respectively, return $A_1^{\mathbf{W}_1, \mathbf{W}_2}$(OUT), where $\mathbf{W}_i(in)$ is evaluated as follows: if $(i, in) \sim (1,$ WO$, x$ **any**$)$, then return $\mathcal{W}(x)$; if $(i, in) \sim (2,$ WO$, x$ **any**$)$, then return $\mathcal{W}'(x)$; and if $(i, in) \sim (2,$ RO$, i$ **int**$, x$ **any**$)$, then return $\mathcal{R}_i(x)$. On input of $(2, x$ **any**$)$ with oracles $\mathcal{W}, \mathcal{W}', \mathcal{R}$, run $A_2^{\mathbf{W}_1, \mathbf{W}_2}(x)$ and return the output. Simulator S is defined as follows. On input of (SETUP), run T(SETUP). On input of (in) with oracle $\mathcal{W}'$, run $T^{\mathbf{W}_2, \mathbf{R}}(in)$, where $\mathbf{W}_2(x)$ is evaluated as $\mathcal{W}'($WO$, x)$ and $\mathbf{R}_i(x)$ is evaluated as $\mathcal{W}'($RO$, i, x)$.

Noting that the runtime of B is $O(t_A)$, the runtime of S is $O(t_T)$, and there exists a $[O(t_A)+f(t_A)]$-time transcript predicate ϕ such that

$$\Pr\big[\,\mathbf{Real}^{\mathsf{out}^\phi}_{W/\vec{R}}(B)\,\big] = \Pr\big[\,\mathbf{Real}^{\mathsf{out}^\psi}_{\hat{W}}(A)\,\big] \quad \text{and} \quad \Pr\big[\,\mathbf{Ref}^{\mathsf{out}^\phi}_{V/\vec{Q}}(B,T)\,\big] = \Pr\big[\,\mathbf{Ref}^{\mathsf{out}^\psi}_{\hat{V}}(A,S)\,\big] \tag{2.1}$$

yields the claim. Predicate ϕ is computed by first rewriting the queries in the transcript in the natural way, then applying ψ to the result. The rewriting step can be done in time linear in the size of the transcript, which, by definition, is linear in the runtime of the adversary.

We now prove claim (2). Adversary B is specified as follows. On input of (SETUP), run $A(\mathsf{SETUP})$. On input of $(1, \mathsf{OUT})$ with oracles $\mathcal{W}, \mathcal{W}'$, run $A_1^{\mathbf{W}_1,\mathbf{W}_2,\mathbf{R}}(\mathsf{OUT})$ and return the output, where $\mathbf{W}_1(x)$ is evaluated as $\mathcal{W}(\mathsf{WO}, x)$, $\mathbf{W}_2(x)$ as $\mathcal{W}'(\mathsf{WO}, x)$, and $\mathbf{R}_i(x)$ as $\mathcal{W}'(\mathsf{RO}, i, x)$. On input of $(2, x\ \mathbf{any})$ with oracles $\mathcal{W}, \mathcal{W}'$, run $A_2^{\mathbf{W}_1,\mathbf{W}_2}(x)$ and return the output. Simulator S is defined as follows. On input of (SETUP), run $T(\mathsf{SETUP})$. On input of (in) with oracles $\mathcal{W}', \mathcal{R}$, run $T^{\mathbf{W}_2}(in)$, where $\mathbf{W}_2(in)$ is evaluated as follows: if $in \sim (\mathsf{WO}, x\ \mathbf{any})$, then return $\mathcal{W}'(x)$; and if $in \sim (\mathsf{RO}, i\ \mathbf{int}, x\ \mathbf{any})$, then return $\mathcal{R}_i(x)$. ■

2.3 The Lifting Lemma

The main technical tool of our framework is its lifting lemma, which states that if $V/\vec{Q}$ is ψ-secure and $W/\vec{R}$ is ψ-indifferentiable from $V/\vec{Q}$, then $W/\vec{R}$ is also ψ-secure. This is a generalization of the main result of MRH, which states that if an object X is secure for a given application and X is indifferentiable from Y, then Y is secure for the same application. In §2.4 we give a precise definition of "application" for which this statement holds.

The Random Oracle Model (ROM). The goal of the lifting lemma is to transform a ψ-attacker against $W/\vec{R}$ into a ψ-attacker against $V/\vec{Q}$. Indifferentiability is used in the following way: given ψ-attacker A and simulator S, we construct a ψ-attacker B and ψ-differentiator D such that, in the real experiment, D outputs 1 if A wins; and in the reference experiment, D outputs 1 if B wins. Adversary B works by running A in the reference experiment with simulator S: intuitively, if the simulation provided by S "looks like" the real experiment, then B should succeed whenever A succeeds.

This argument might seem familiar, even to readers who have no exposure to the notion of indifferentiability. Indeed, a number of reductions in the provable security literature share the same basic structure. For example, when proving a signature scheme is unforgeable under chosen message attack (UF-CMA), the first step is usually to transform the attacker into a weaker one that does not have access to a signing oracle. This argument involves exhibiting a simulator that correctly answers the UF-CMA adversary's signing-oracle queries using only the public key (cf. [32, Theorem 4.1]): if the simulation is efficient, then we can argue that security in the weak attack model reduces to UF-CMA. Similarly, to prove a public-key encryption (PKE) scheme is indistinguishable under chosen ciphertext attack (IND-CCA), the strategy might be to exhibit a simulator for the decryption oracle in order to argue that IND-CPA reduces to IND-CCA.

Given the kinds of objects we wish to study, it will be useful for us to accommodate these types of arguments in the lifting lemma. In particular, Lemma 1 considers the case in which one of the resources in the reference experiment is an RO that may be "programmed" by the simulator. (As we discuss in §2.4.2, this capability is commonly used when simulating signing-oracle queries.) In our setting, the RO is programmed by passing it an object M via its SET-operator (2.1:9-14), which is run by the RO in order to populate the table. Normally we will require M to be an entropy source with the following properties.

Definition 6 (Sources)**.** Let $\mu, \rho \geq 0$ be real numbers and $\mathcal{X}, \mathcal{Y}$ be computable sets. An *$\mathcal{X}$-source* is a stateless object that exports a $(\,)$-$\mathbf{elem}_{\mathcal{X}}$-operator. An *$(\mathcal{X}, \mathcal{Y})$-source* is a stateless object that exports a $(\,)$-$(\mathbf{elem}_{\mathcal{X}\times\mathcal{Y}}, \mathbf{any})$-operator. Let M be an $(\mathcal{X}, \mathcal{Y})$-source and let $((X, Y), \Sigma)$ be random variables distributed according to M. (That is, run $((x, y), \sigma) \leftarrow M(\,)$ and assign $X \leftarrow x$, $Y \leftarrow y$, and $\Sigma \leftarrow \sigma$.) We say that M is *(μ, ρ)-min-entropy* if the following conditions hold:

(1) For all x and y it holds that $\Pr\big[\,X = x\,\big] \leq 2^{-\mu}$ and $\Pr\big[\,Y = y\,\big] \leq 2^{-\rho}$.

(2) For all y and σ it holds that $\Pr\big[\,Y = y\,\big] = \Pr\big[\,Y = y \mid \Sigma = \sigma\,\big]$.

We refer to σ as the *auxiliary information* (cf. "source" in [23, §3]). □

A brief explanation is in order. When a source is executed by an RO, the table T is programmed with the output point (x, y) so that $T[x] = y$. The auxiliary information σ is returned to the caller (2.1:14), allowing the source to provide the simulator a "hint" about how the point was chosen. Condition (1) is our min-entropy requirement for sources. We also require condition (2), which states that the range point programmed by the source is independent of the auxiliary information.

Definition 7 (The ROM)**.** Let $\mathcal{X}, \mathcal{Y}$ be computable sets where $\mathcal{Y}$ is finite, let $q, p \geq 0$ be integers, and let $\mu, \rho \geq 0$ be real numbers. A *random oracle from $\mathcal{X}$ to $\mathcal{Y}$ with query limit* (q, p) is the object $R = \mathbf{Ro}(\mathcal{X}, \mathcal{Y}, q, p)$ specified in Figure 2.1. This object permits at most q unique RO queries and at most p RO-programming queries. If the query limit is unspecified, then it is $(\infty, 0)$ so that the object permits any number of RO queries but no RO-programming queries. Objects program the RO by making queries matching the pattern $(\mathsf{SET}, M\ \mathbf{object})$. An object that makes such queries is called (μ, ρ)-$(\mathcal{X}, \mathcal{Y})$-*min-entropy* if, for all such queries, the object M is always a (μ, ρ)-min-entropy $(\mathcal{X}, \mathcal{Y})$-source. An object that makes no queries matching this pattern is *not RO-programming (n.r.)*. □

To model a function H as a random oracle in an experiment, we revise the experiment by replacing each call of the form "$H(\cdots)$" with a call of the form "$\mathcal{R}_i(\cdots)$", where i is the index of the RO in the shared resources of the experiment, and $\mathcal{R}$ is the name of the resource oracle. When specifying a cryptographic scheme whose security analysis is in the ROM, we will usually skip this rewriting step and simply write the specification in terms of $\mathcal{R}_i$-queries: to obtain the standard model experiment, one would instantiate the i-th resource with H instead of an RO.

Lemma 1 (Lifting)**.** *Let $\vec{I} = (I_1, \ldots, I_u), \vec{J} = (J_1, \ldots, J_v)$ be resources; let $\mathcal{X}, \mathcal{Y}$ be computable sets, where $\mathcal{Y}$ is finite; let $N = |\mathcal{Y}|$; let $\mu, \rho \geq 0$ be real numbers for which $\log N \geq \rho$; let $q, p \geq 0$ be integers; let R and P be random oracles for $\mathcal{X}, \mathcal{Y}$ with query limits $(q + p, 0)$ and (q, p) respectively; let W, V be n.r. worlds; and let ψ be a transcript predicate. For every t_A-time, (a_1, a_2, a_r)-query, n.d. adversary A and t_S-time, (s_2, s_r)-query, (μ, ρ)-$(\mathcal{X}, \mathcal{Y})$-min-entropy simulator S, there exist n.d. adversaries D and B for which*

$$\mathbf{Adv}^{\mathrm{main}^{\psi}}_{W/\vec{J}}(A) \leq \mathit{\Delta} + \mathbf{Adv}^{\mathrm{main}^{\psi}}_{V/\vec{I}\,.\,R}(B) + \mathbf{Adv}^{\mathrm{sr\text{-}indiff}^{\psi}}_{W/\vec{J}, V/\vec{I}\,.\,P}(D, S)\,,$$

where $\mathit{\Delta} = p\left[(p + q)/2^{-\mu} + \sqrt{N/2^{\rho} \cdot \log(N/2^{\rho})}\right]$, D is $O(t_A)$-time and $(a_1{+}1, a_2, a_r)$-query, and B is $O(t_A t_S)$-time and $(a_1, a_2 s_2, (a_2 + a_r)s_r)$-query.

Apart from dealing with RO programmability, which accounts for the $\mathit{\Delta}$-term in the bound, the proof is essentially the same argument as the sufficient condition in [104, Theorem 1] (cf. [129, Theorem 1]). The high min-entropy of domain points programmed by the simulator ensures that RO-programming queries are unlikely to collide with standard RO queries. However, we will need that range points are statistically close to uniform; otherwise the $\mathit{\Delta}$-term becomes vacuous. Note that $\mathit{\Delta} = 0$ whenever programming is disallowed.

Proof of Lemma 1. Let $D = \mathbf{D}(A)$ as specified in Figure 2.5. This adversary works by executing A with access to its oracles and outputting 1 if A wins (that is, the output of $\mathcal{W}(\mathsf{WIN})$ on line 2.5:5 is 1). Then

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{main}^{\psi}}_{W/\vec{J}}(A) &= \Pr\big[\,\mathbf{Real}^{\mathsf{win}^{\psi}}_{W/\vec{J}}(A)\,\big] + \Big(\Pr\big[\,\mathbf{Ref}^{\mathsf{win}^{\psi}}_{V/\vec{I}\,.\,P}(A, S)\,\big] - \Pr\big[\,\mathbf{Ref}^{\mathsf{win}^{\psi}}_{V/\vec{I}\,.\,P}(A, S)\,\big]\Big) && (2.2)\\
&= \Big(\Pr\big[\,\mathbf{Real}^{\mathsf{out}^{\psi}}_{W/\vec{J}}(D)\,\big] - \Pr\big[\,\mathbf{Ref}^{\mathsf{out}^{\psi}}_{V/\vec{I}\,.\,P}(D, S)\,\big]\Big) + \Pr\big[\,\mathbf{Ref}^{\mathsf{win}^{\psi}}_{V/\vec{I}\,.\,P}(A, S)\,\big] && (2.3)\\
&= \mathbf{Adv}^{\mathrm{sr\text{-}indiff}^{\psi}}_{W/\vec{J}, V/\vec{I}\,.\,P}(D, S) + \Pr\big[\,\mathbf{Ref}^{\mathsf{win}^{\psi}}_{V/\vec{I}\,.\,P}(A, S)\,\big]\,. && (2.4)
\end{aligned}$$

Our goal for the remainder is to construct a MAIN$^{\psi}$-adversary B from A and S whose advantage upper-bounds the last term on the right hand side of Eq. (2.4). The main difficulty is that S might try to program R, which is not allowed because R has query limit $(q + p, 0)$. Our solution is to use the min-entropy of S in order transition into a world in which its RO-programming queries are answered by standard RO queries.

Let R^{**} be the object defined just like P except that it answers queries matching $(\mathsf{SET}, M\ \mathbf{object})$ as follows. If $j \geq p$ then immediately halt and return $\perp$. Otherwise run $((x, \_), \sigma) \leftarrow M(\,)$ and increment j. If $T[x]$ is undefined,

```
spec $\mathbf{D}$:
  // In real: $\mathcal{W}$ points to $W_1$; $\mathcal{W}'$ to $W_2$; $\mathcal{R}$ to resources
  // In ref: $\mathcal{W}$ points to $V_1$; $\mathcal{W}'$ to $V_2$; $\mathcal{R}$ to resources
  var $A$ object
  op (SETUP): $A$(SETUP)
  op$^{\mathcal{W},\mathcal{W}',\mathcal{R}}$ (1, OUT): $A_1^{\mathcal{W},\mathcal{W}',\mathcal{R}}$(OUT); ret $\mathcal{W}$(WIN)
  op$^{\mathcal{W},\mathcal{W}',\mathcal{R}}$ (2, $x$ any): $A_2^{\mathcal{W},\mathcal{W}',\mathcal{R}}(x)$

spec $\mathbf{B}$: // $\mathcal{V}$ points to $V_1$; $\mathcal{V}'$ to $V_2$; $\mathcal{R}$ to resources
  var $A, S$ object
  op (SETUP): $S$(SETUP); $A$(SETUP)
  op$^{\mathcal{V},\mathcal{V}',\mathcal{R}}$ (1, OUT): ret $A_1^{\mathcal{V},\mathbf{S}_2,\mathbf{S}_3}$(OUT)
  op$^{\mathcal{V},\mathcal{V}',\mathcal{R}}$ (2, $x$ any): ret $A_2^{\mathcal{V},\mathbf{S}_2,\mathbf{S}_3}(x)$

interface $\mathbf{R}$:
  op ($u$+1, (SET, $M$ object)):
      $((x, \_), \sigma) \leftarrow M(\,)$; ret $((x, \mathcal{R}_{u+1}(x)), \sigma)$
  op ($i$ int, $x$ any): ret $\mathcal{R}_i(x)$

procedure $\mathbf{S}(i, x)$:
  ret $S_i^{\mathcal{V}',\mathbf{R}}(x)$
```

```
spec $\mathbf{Ro}^*$:
  var $\mathcal{X}, \mathcal{Y}$ set, $a$ bool, $t, q, p$ int
  var $T, U, V$ table
  var $i, j$ int, $bad$ bool
  op (SETUP):
    $i, j \leftarrow 1$; $bad \leftarrow 0$
    $T, U, V \leftarrow [\,]$
  op ($x$ elem$_{\mathcal{G}}$):
    if $i \geq q$ then ret $\perp$
    if $T[x] = \perp$ then
      $i \leftarrow i + 1$; $T[x] \twoheadleftarrow \mathcal{Y}$
    ret $T[x]$
  op (SET, $M$ object):
    if $j \geq p$ then ret $\perp$
    $j \leftarrow j + 1$; $((x, y), \sigma) \leftarrow M(\,)$
    if $T[x] \neq \perp$ then $bad \leftarrow 1$
      if $a = 0$ then ret $((x, T[x]), \sigma)$
    $U_j \twoheadleftarrow \mathcal{Y}$; $V_j \leftarrow y$
    if $j \leq t$ then $T[x] \leftarrow U_j$
    else $T[x] \leftarrow V_j$
    ret $((x, T[x]), \sigma)$
```

Figure 2.5: Reductions and hybrid experiment RO used in the proof of Lemma 1.

then set $T[x] \twoheadleftarrow \mathcal{Y}$ exactly as the standard, call-for-value operator does. Finally, return $((x, T[x]), \sigma)$. Next we will show that

$$\Pr\big[\,\mathbf{Ref}^{\mathsf{win}^\psi}_{V/\vec{I}\,.\,P}(A, S)\,\big] \leq \Delta + \Pr\big[\,\mathbf{Ref}^{\mathsf{win}^\psi}_{V/\vec{I}\,.\,R^{**}}(A, S)\,\big]\,. \tag{2.5}$$

For each $a \in \{0, 1\}$ and $t \in [0..p]$ let $R^*_{a,t} = \mathbf{Ro}^*(\mathcal{X}, \mathcal{Y}, a, t, q, p)$ as specified in Figure 2.5. Object $R^*_{a,t}$ works like P except that the behavior of SET-queries depends on the parameters a, t. The first parameter, a, determines if SET-queries overwrite key/value pairs already in the table. If $a = 1$, then when attempting to set a point (x, y) in the table T, if $T[x]$ is already defined then the value of $T[x]$ will overwritten; otherwise $T[x]$ stays the same. Thus, $a = 1$ corresponds to the usual operation of programming queries, whereas $a = 0$ changes this behavior. The second parameter, t, changes the distribution of values entered into the table as follows: each call matching (SET, M **object**) following the t-th runs $((x, y), \sigma) \leftarrow M(\,)$ and sets $T[x] \leftarrow y$ in the usual manner for programming queries; for every call preceding and including the t-th, the value of $T[x]$ is sampled uniformly from $\mathcal{Y}$.

Define the random variable $\mathsf{Succ}_{a,t}$ to be the outcome of $\mathbf{Ref}^{\mathsf{win}^\psi}_{V/\vec{I}\,.\,R^*_{a,t}}(A, S)$ for each a and t. The behavior of $R^*_{1,0}$ is identical to P; and the behavior of $R^*_{0,p}$ is identical to R^{**}. Hence,

$$\Pr\big[\,\mathsf{Succ}_{1,0}\,\big] = \Pr\big[\,\mathbf{Ref}^{\mathsf{win}^\psi}_{V/\vec{I}\,.\,P}(A, S)\,\big] \tag{2.6}$$

and

$$\Pr\big[\,\mathsf{Succ}_{0,p}\,\big] = \Pr\big[\,\mathbf{Ref}^{\mathsf{win}^\psi}_{V,\vec{I}\,.\,R^{**}}(A, S)\,\big]\,. \tag{2.7}$$

Our objective is to bound the probability of $\mathsf{Succ}_{1,0}$ as a function of the probability of $\mathsf{Succ}_{0,p}$. First, observe that for a given t, the outputs of calls to $R^*_{0,t}$ and $R^*_{1,t}$ are identically distributed until the *bad* flag gets set. By the fundamental lemma of game playing [33] we have that

$$\begin{aligned}
\Pr\big[\,\mathsf{Succ}_{1,0}\,\big] &\leq \Pr\big[\,\mathsf{Succ}_{0,0}\,\big] + \Pr\big[\,\mathbf{Ref}^{\mathsf{win}^\psi}_{V/\vec{I}\,.\,R^*_{1,0}}(A, S)\;\mathbf{sets}\;R^*_{1,0}.bad\,\big] && (2.8)\\
&\leq \Pr\big[\,\mathsf{Succ}_{0,0}\,\big] + \frac{p(p+q)}{2^\mu}\,. && (2.9)
\end{aligned}$$

Eq. (2.9) follows from the assumption that S is (μ, ρ)-$(\mathcal{X}, \mathcal{Y})$-min-entropy: the last term upper-bounds the probability that a programmed point collides with an existing point in the table. Next, observe that

$$\begin{aligned}
\Pr\left[\,\mathsf{Succ}_{0,0}\,\right] &= \Pr\left[\,\mathsf{Succ}_{0,0}\,\right] + \sum_{t=1}^{p} \left(\Pr\left[\,\mathsf{Succ}_{0,t}\,\right] - \Pr\left[\,\mathsf{Succ}_{0,t}\,\right]\right) && (2.10)\\
\Pr\left[\,\mathsf{Succ}_{0,0}\,\right] - \Pr\left[\,\mathsf{Succ}_{0,p}\,\right] &= \sum_{t=0}^{p-1} \left(\Pr\left[\,\mathsf{Succ}_{0,t}\,\right] - \Pr\left[\,\mathsf{Succ}_{0,t+1}\,\right]\right) && (2.11)\\
&\leq \delta p\,, && (2.12)
\end{aligned}$$

where $\delta := \max_{0\leq t\leq p-1} \delta_t$ and $\delta_t := \left|\Pr\left[\,\mathsf{Succ}_{0,t}\,\right] - \Pr\left[\,\mathsf{Succ}_{0,t+1}\,\right]\right|$. Let $U_t = y$ denote the event that

$$\mathbf{Ref}^{\mathsf{win}^{\psi}}_{V/\vec{I}\,.\,R^*_{0,t}}(A, S)\,\mathbf{sets}\,R^*_{0,t}.U_t = y$$

for each $y \in \mathcal{Y}$, and define $V_t = y$ in kind. We can use the statistical distance between U_t and V_t to upper-bound δ_t. In particular, we claim that

$$\delta_t \;\leq\; \frac{1}{2}\sum_{y\in\mathcal{Y}} \left|\Pr\left[\,V_t = y\,\right] - \Pr\left[\,U_t = y\,\right]\right| \qquad (2.13)$$

for all $t \in [0..p-1]$. More generally, we have the following.

Claim 1. Let $\mathcal{X}$ be a computable set and let X and Y be random variables with support $\mathcal{X}$. For every halting object D it holds that $\Pr\left[\,D(X)\,\right] - \Pr\left[\,D(Y)\,\right] \leq 1/2\sum_{x\in\mathcal{X}} \left|\Pr\left[\,X = x\,\right] - \Pr\left[\,Y = y\,\right]\right|$.

Proof.[1] Without loss of generality, we may rewrite D as a functional object F for which there is a set $\mathcal{R}$ and $\mathcal{R}$-source Ω such that $\Pr\left[\,D(X) = 1\,\right] = \Pr\left[\,R \leftarrow \Omega(\,) : F(X, R) = 1\,\right]$. Let R and S be independent random variables distributed according to Ω, let $B_1 = (X, R)$, and let $B_0 = (Y, S)$. Let $\mathcal{V} = \mathcal{X} \times \mathcal{R}$ denote the support of B_1 and B_0. Then

$$\begin{aligned}
\Pr\left[\,D(X)\,\right] - \Pr\left[\,D(Y)\,\right] &\leq \left|\Pr\left[\,F(B_1)\,\right] - \Pr\left[\,F(B_0)\,\right]\right| && (2.14)\\
&\leq \max_{f:\mathcal{V}\to\{0,1\}} \left|\Pr\left[\,f(B_1) = 1\,\right] - \Pr\left[\,f(B_0) = 1\,\right]\right| && (2.15)\\
&\leq \max_{\mathcal{W}\subseteq\mathcal{V}} \left|\Pr\left[\,B_1 \in \mathcal{W}\,\right] - \Pr\left[\,B_0 \in \mathcal{W}\,\right]\right| && (2.16)\\
&= \left|\Pr\left[\,B_1 \in \mathcal{W}^*\,\right] - \Pr\left[\,B_0 \in \mathcal{W}^*\,\right]\right|\,, && (2.17)
\end{aligned}$$

where $\mathcal{W}^*$ denotes the subset of $\mathcal{V}$ that maximizes the quantity on the right hand side of Eq. (2.17), and Eq. (2.16) is obtained by writing each f as a predicate $f(v) \mapsto v \in \mathcal{W}$ for some $\mathcal{W} \subseteq \mathcal{V}$.

Let $\xi := \left|\Pr\left[\,B_1 \in \mathcal{W}^*\,\right] - \Pr\left[\,B_0 \in \mathcal{W}^*\,\right]\right|$. Note that either $\mathcal{W}^* = \mathcal{T}$ or $\mathcal{W}^* = \mathcal{V} \setminus \mathcal{T}$, where $\mathcal{T} := \{v \in \mathcal{V} : \Pr\left[\,B_1 = v\,\right] - \Pr\left[\,B_0 = v\,\right] \geq 0\}$. But, since

$$\Pr\left[\,B_1 \in \mathcal{T}\,\right] + \Pr\left[\,B_1 \in \mathcal{V}\setminus\mathcal{T}\,\right] = \Pr\left[\,B_0 \in \mathcal{T}\,\right] + \Pr\left[\,B_0 \in \mathcal{V}\setminus\mathcal{T}\,\right] \qquad (2.18)$$

by the law of total probability, we have that

$$\begin{aligned}
\Pr\left[\,B_1 \in \mathcal{T}\,\right] - \Pr\left[\,B_0 \in \mathcal{T}\,\right] &= \Pr\left[\,B_0 \in \mathcal{V}\setminus\mathcal{T}\,\right] - \Pr\left[\,B_1 \in \mathcal{V}\setminus\mathcal{T}\,\right] && (2.19)\\
\left|\Pr\left[\,B_1 \in \mathcal{T}\,\right] - \Pr\left[\,B_0 \in \mathcal{T}\,\right]\right| &= \left|\Pr\left[\,B_1 \in \mathcal{V}\setminus\mathcal{T}\,\right] - \Pr\left[\,B_0 \in \mathcal{V}\setminus\mathcal{T}\,\right]\right| && (2.20)\\
\xi &= \left|\Pr\left[\,B_1 \in \mathcal{T}\,\right] - \Pr\left[\,B_0 \in \mathcal{T}\,\right]\right|\,. && (2.21)
\end{aligned}$$

By Eq. (2.18) again,

$$\begin{aligned}
\xi &= \left|\Pr\left[\,B_1 \in \mathcal{T}\,\right] - \Pr\left[\,B_0 \in \mathcal{T}\,\right]\right| && (2.22)\\
&= \frac{1}{2}\Big(\left|\Pr\left[\,B_1 \in \mathcal{T}\,\right] - \Pr\left[\,B_0 \in \mathcal{T}\,\right]\right| + \left|\Pr\left[\,B_1 \in \mathcal{V}\setminus\mathcal{T}\,\right] - \Pr\left[\,B_0 \in \mathcal{V}\setminus\mathcal{T}\,\right]\right|\Big) && (2.23)
\end{aligned}$$

[1] The following argument is adapted from Daniel Wichs' lecture notes.

$$
\begin{aligned}
&= \frac{1}{2}\left(\sum_{v\in\mathcal{T}}\left|\Pr\left[\,B_1 = v\,\right] - \Pr\left[\,B_0 = v\,\right]\right| + \sum_{v\in\mathcal{V}\setminus\mathcal{T}}\left|\Pr\left[\,B_1 = v\,\right] - \Pr\left[\,B_0 = v\,\right]\right|\right) && (2.24)\\
&= \frac{1}{2}\sum_{v\in\mathcal{V}}\left|\Pr\left[\,B_1 = v\,\right] - \Pr\left[\,B_0 = v\,\right]\right| . && (2.25)
\end{aligned}
$$

Since X, R (resp. Y, S) are independently distributed and R, S are i.i.d.,

$$
\begin{aligned}
\xi &= \frac{1}{2}\sum_{(z,r)\in\mathcal{V}}\left|\Pr\left[\,X = z\,\right]\Pr\left[\,R = r\,\right] - \Pr\left[\,Y = z\,\right]\Pr\left[\,S = r\,\right]\right| && (2.26)\\
&= \frac{1}{2}\sum_{(z,r)\in\mathcal{V}}\left|\Pr\left[\,R = r\,\right]\cdot\left(\Pr\left[\,X = z\,\right] - \Pr\left[\,Y = z\,\right]\right)\right| && (2.27)\\
&= \frac{1}{2}\sum_{z\in\mathcal{X}}\sum_{r\in\mathcal{R}}\Pr\left[\,R = r\,\right]\cdot\left|\Pr\left[\,X = z\,\right] - \Pr\left[\,Y = z\,\right]\right| && (2.28)\\
&= \frac{1}{2}\sum_{z\in\mathcal{X}}\left|\Pr\left[\,X = z\,\right] - \Pr\left[\,Y = z\,\right]\right| . && (2.29)
\end{aligned}
$$

This concludes the proof of Claim 1. □

We can obtain a closed form for δ_t using Kullback-Leibler divergence [63]. Since $\Pr\left[\,U_t = y\,\right] = 0$ implies $\Pr\left[\,V_t = y\,\right] = 0$ for all $y \in \mathcal{Y}$, and since S is (μ,ρ)-$(\mathcal{X},\mathcal{Y})$-min-entropy, we have that

$$
\begin{aligned}
\delta_t &\le \frac{1}{2}\sum_{y\in\mathcal{Y}}\left|\Pr\left[\,V_t = y\,\right] - \Pr\left[\,U_t = y\,\right]\right| && (2.30)\\
&\le \sqrt{\frac{1}{2}\sum_{y\in\mathcal{Y}}\Pr\left[\,V_t = y\,\right]\cdot\log\left(\frac{\Pr\left[\,V_t = y\,\right]}{\Pr\left[\,U_t = y\,\right]}\right)} && (2.31)\\
&\le \sqrt{\frac{1}{2}\sum_{y\in\mathcal{Y}}2^{-\rho}\cdot\log\left(\frac{2^{-\rho}}{N^{-1}}\right)} && (2.32)\\
&= \sqrt{N/2^{\rho+1}\cdot\log(N/2^{\rho})}\,. && (2.33)
\end{aligned}
$$

Thus, $\delta \le \sqrt{N/2^{\rho}\cdot\log(N/2^{\rho})}$. Summarizing, we have

$$
\begin{aligned}
\Pr\left[\,\mathbf{Ref}^{\mathsf{win}^{\psi}}_{V/\vec{I}\,.\,P}(A,S)\,\right] &= \left(\Pr\left[\,\mathsf{Succ}_{1,0}\,\right] - \Pr\left[\,\mathsf{Succ}_{0,0}\,\right]\right) + \Pr\left[\,\mathsf{Succ}_{0,0}\,\right] && (2.34)\\
&\le p(p+q)/2^{-\mu} + \Pr\left[\,\mathsf{Succ}_{0,0}\,\right] && (2.35)\\
&= p(p+q)/2^{-\mu} + \left(\Pr\left[\,\mathsf{Succ}_{0,0}\,\right] - \Pr\left[\,\mathsf{Succ}_{0,p}\,\right]\right) + \Pr\left[\,\mathsf{Succ}_{0,p}\,\right] && (2.36)\\
&\le p(p+q)/2^{-\mu} + \delta p + \Pr\left[\,\mathsf{Succ}_{0,p}\,\right] && (2.37)\\
&\le p\left[(p+q)/2^{-\mu} + \sqrt{N/2^{\rho}\cdot\log(N/2^{\rho})}\right] + \Pr\left[\,\mathbf{Ref}^{\mathsf{win}^{\psi}}_{V,\vec{I}\,.\,R^{**}}(A,S)\,\right] . && (2.38)
\end{aligned}
$$

The final step is to bound the last term on the right hand side of Eq. (2.38). Observe that SET-operator queries to R^{**} can be simulated using the standard RO operator. In particular, let $B = \mathbf{B}(A, S)$ as specified in Figure 2.5. Adversary B runs A, answering its queries as follows. Queries to the main interface are forwarded to B's $\mathcal{V}$ oracle; and queries to the auxiliary and resource interfaces are forwarded to the simulator S, which is run with B's $\mathcal{V}'$ and $\mathcal{R}$ oracles except that queries to $\mathcal{R}$ matching $(u{+}1, (\mathsf{SET}, M\ \textbf{object}))$ are transformed into $(u{+}1, x)$ queries, as shown on line 2.5:12. (We call interface $\mathbf{R}$ a *pure interface*: it behaves much like a procedure, except that the input is matched to one of its operators. Its syntax is given by term interface in Figure 2.1.) Since R has query limit $q+p, 0$, it follows that

$$
\Pr\left[\,\mathbf{Ref}^{\mathsf{win}^{\psi}}_{V/\vec{I}\,.\,R^{**}}(A,S)\,\right] = \Pr\left[\,\mathbf{Real}^{\mathsf{win}^{\psi}}_{V/\vec{I}\,.\,R}(B)\,\right] . \qquad (2.39)
$$

To finish the proof, we need only to account for the resources of D and B. The runtime of D is $O(t_A + t_W)$, where t_W is the runtime of W, since it involves running A and the WIN-operator of W_1 once. But $t_W \le t_A$ because, by

```
spec Wo: // $\mathcal{R}$ points to resources; $\mathcal{A}$ to $A_2$
  1 var $G, X$ object
  2 op (SETUP): $G$(SETUP); $X$(SETUP)
  3 op (1, WIN): ret $G_1$(WIN)
  4 op$^{\mathcal{A},\mathcal{R}}$ (1, $x$ any): ret $\mathbf{G}_1(x)$
  5 op$^{\mathcal{A},\mathcal{R}}$ (2, $x$ any): ret $\mathbf{X}_2(x)$

procedure $\mathbf{X}(i, x)$:
  6 ret $X_i^{\mathbf{G}_2,\mathcal{R}}(x)$

procedure $\mathbf{G}(i, x)$:
  7 ret $G_i^{\mathbf{X}_1,\mathcal{A}}(x)$
```

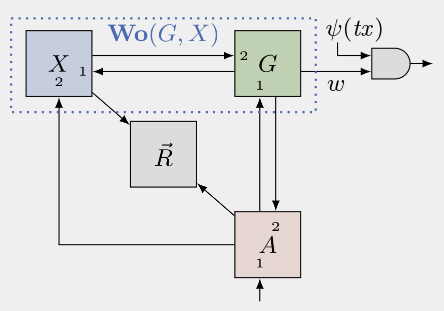

Figure 2.6: Left: Specification **Wo** for building a world from a security game G and system X. Right: Who may call whom in experiment $\mathbf{Real}^{\Phi}_{W/\vec{R}}(A)$, where $W = \mathbf{Wo}(G, X)$.

convention, the runtime of A includes the time required to evaluate its queries. Hence, the runtime of D is simply $O(t_A)$. The runtime of B is $O(t_A + (q_2 + q_r) \cdot t_S)$ if A makes at most q_2 queries to the auxiliary interface and at most q_r to the resource. Since $q_2 + q_r \leq t_A$ by convention (the adversary's runtime includes the time needed to evaluate its queries), it follows that the runtime of B is $O(t_A + t_A t_S) = O(2 t_A t_S) = O(t_A t_S)$. ■

2.4 Games and the Preservation Lemma

Lemma 1 says that indifferentiability of world W from world V means that security of V implies security of W. This starting point is more general than the usual one, which is to first argue indifferentiability of some system X from another system Y, then use the composition theorem of MRH in order to argue that security of Y for some application implies security of X for the same application. Here we formalize the same kind of argument by specifying the construction of a world from a system X and a game G that defines the system's security.

A *game* is a halting object that exports a functional (1, WIN)-**bool**-operator. A *system* is a halting object. Figure 2.6 specifies the composition of a game G and system X into a world $W = \mathbf{Wo}(G, X)$ in which the adversary interacts with G's main interface and X's auxiliary interface, and G interacts with X's main interface. The system X makes queries to G's auxiliary interface, and G in turn makes queries to the adversary's auxiliary interface. As shown in right hand side of Figure 2.6, it is the game that decides whether the adversary has won: when the real experiment calls W_1(WIN) on line 2.3:4, this call is answered by the operator defined by **Wo** on line 2.6:3, which returns G_1(WIN).

Definition 8 (G^{ψ} security)**.** Let ψ be a transcript predicate, G be a game, X be a system, $\vec{R}$ be resources, and A be an adversary. We define the G^{ψ} advantage of A in attacking $X/\vec{R}$ as

$$\mathbf{Adv}^{G^{\psi}}_{X/\vec{R}}(A) := \mathbf{Adv}^{\mathrm{main}^{\psi}}_{\mathbf{Wo}(G,X)/\vec{R}}(A)\,.$$

We write $\mathbf{Adv}^{G}_{X/\vec{R}}(A)$ whenever $\psi(tx) = 1$ for all tx. Informally, we say that $X/\vec{R}$ is G^{ψ}-secure if the G^{ψ} advantage of any efficient adversary is small. □

World **Wo** formalizes the class of systems for which we will define security in this dissertation. While the execution semantics of games and systems seems quite natural, we remark that other ways of capturing security notions are possible. We are restricted only by the execution semantics of the real experiment (Def. 1). Indeed, there are natural classes of security definitions we cannot capture, including those described by multi-stage games [129].

For our particular class of security notions we can prove the following useful lemma. Intuitively, the "preservation" lemma below states that if a system X is ψ-indifferentiable from Y, then $\mathbf{Wo}(G, X)$ is ψ-indifferentiable from $\mathbf{Wo}(G, Y)$ for any game G.

Lemma 2 (Preservation)**.** *Let ψ be a transcript predicate, X, Y be objects, and $\vec{R}, \vec{Q}$ be resources. For every $(g_1, \_)$-query game G, t_A-time, (a_1, a_2, a_r)-query, n.d. adversary A, and simulator S there exists an n.d. adversary B such that*

$$\mathbf{Adv}^{\text{sr-indiff}^{\psi}}_{W/\vec{R},V/\vec{Q}}(A, S) \leq \mathbf{Adv}^{\text{sr-indiff}^{\psi}}_{X/\vec{R},Y/\vec{Q}}(B, S)\,,$$

where $W = \mathbf{Wo}(G, X)$, $V = \mathbf{Wo}(G, Y)$, and B is $O(t_A)$-time and $(a_1 g_1, a_2, a_r)$-query.

Proof. Adversary B simulates the execution of A in its experiment as follows. On input of (SETUP), run $G(\mathsf{SETUP})$ and $A(\mathsf{SETUP})$. On input of $(1, \mathsf{OUT})$ with oracles $\mathcal{X}, \mathcal{X}', \mathcal{R}$, return $\mathbf{A}_1(\mathsf{OUT})$, where $\mathbf{A}(i, x) = A_i^{\mathbf{G}_1, \mathcal{X}', \mathcal{R}}(x)$ and $\mathbf{G}(i, x) = G_i^{\mathcal{X}, \mathbf{A}_2}(x)$. On on input of $(2, x\ \textbf{any})$ with oracles $\mathcal{X}, \mathcal{X}', \mathcal{R}$, return $\mathbf{G}_2(x)$. By construction we have that

$$\Pr\big[\,\mathbf{Real}^{\mathsf{out}^{\psi}}_{X/\vec{R}}(B)\,\big] = \Pr\big[\,\mathbf{Real}^{\mathsf{out}^{\psi}}_{\mathbf{Wo}(G,X)/\vec{R}}(A)\,\big] \tag{2.40}$$

and

$$\Pr\big[\,\mathbf{Ref}^{\mathsf{out}^{\psi}}_{Y/\vec{Q}}(B, S)\,\big] = \Pr\big[\,\mathbf{Ref}^{\mathsf{out}^{\psi}}_{\mathbf{Wo}(G,Y)/\vec{Q}}(A, S)\,\big]\,. \tag{2.41}$$

The runtime of B is $O(t_A + t_G)$ since B runs $G(\mathsf{SETUP})$ on input of (SETUP). But because A's runtime includes the time needed to evaluate its oracle queries (i.e., evaluate calls to G), we have that B is $O(t_A)$-time. ■

2.4.1 The MRH Composition Theorem

In the remainder, we will use the preservation lemma in the following way. Given that some reference system Y is G^{ψ}-secure, we wish to know if the corresponding real system X is also G^{ψ}-secure. We first apply Lemma 1, reducing security to the ψ-indifferentiability of $\mathbf{Wo}(G, X)$ from $\mathbf{Wo}(G, Y)$. We then apply Lemma 2, reducing security to the ψ-indifferentiability of X from Y. This yields the composition theorem of MRH, but formulated for objects instead of interacting systems.

Proposition 2 (Analogue of [104, Theorem 1])**.** *The following conditions hold for all systems X, Y.*

(1) (Indifferentiability is sufficient.) *For every t_G-time game G, t_A-time, n.d. adversary A, and t_S-time simulator S there exist n.d. adversaries D and B for which $\mathbf{Adv}^{G}_{X}(A) \leq \mathbf{Adv}^{G}_{Y}(B) + \mathbf{Adv}^{\text{indiff}}_{X,Y}(D, S)$, where D is $O(t_A)$-time, and B is $O(t_A t_S)$-time.*

(2) (Indifferentiability is necessary.) *For every t_B-time adversary B, t_D-time adversary D there exist an adversary A, simulator S, and game $\hat{G}$ for which $\mathbf{Adv}^{\text{indiff}}_{X,Y}(D, S) + \mathbf{Adv}^{\hat{G}}_{Y}(B) \leq \mathbf{Adv}^{\hat{G}}_{X}(A)$, where A is $O(t_D)$-time, S is $O(t_B)$-time, and $\hat{G}$ is $O(t_D)$-time.*

Proof. Claim (1) is a corollary of Lemma 1 (letting $q, p = 0$ and $\vec{I}, \vec{J} = (\,)$) and Lemma 2. We address the necessary condition (claim (2)) in the remainder. Let $A = \mathbf{A}(\,)$ and $\hat{G} = \hat{\mathbf{G}}(D)$ as specified in Figure 2.7. The world $\hat{G}$ works by running D, answering its oracle queries using its own oracles. As shown in Figure 2.6, world $\hat{G}$ is given an oracle for X_1 in the real experiment, Y_1 in the reference experiment, and A_2 in both experiments.

Adversary A works by running D (via a query to $\hat{G}$), relaying D's $\mathcal{A}$-queries to its own $\mathcal{X}$-oracle. (See Figure 2.7 for an illustration.) Observe that A wins in the MAIN experiment exactly when $\mathbf{Real}^{\mathsf{out}}_{X}(D) = 1$. Thus,

$$\mathbf{Adv}^{\text{main}}_{\mathbf{Wo}(\hat{G},X)}(A) = \Pr\big[\,\mathbf{Real}^{\mathsf{out}}_{X}(D)\,\big]. \tag{2.42}$$

Now consider the MAIN advantage of B in attacking $\mathbf{Wo}(\hat{G}, Y)$. Adversary B wins only if D outputs 1, which occurs only if B asks (OUT) of its world's main interface. In this case, B wins precisely when $\mathbf{Ref}^{\mathsf{out}}_{Y}(D, S) = 1$, where $S = B$. Then

$$\mathbf{Adv}^{\text{main}}_{\mathbf{Wo}(\hat{G},Y)}(B) \leq \Pr\big[\,\mathbf{Ref}^{\mathsf{out}}_{Y}(D, S)\,\big] \tag{2.43}$$

and the claim follows. ■

2.4.2 Exercise: Joint Security of Signing and Encryption

Let us consider a simple exercise that illustrates how our framework is used. Suppose we are given a scheme $\mathit{\Pi}$ with public/secret key pair $(PK = g^s, s) \in \mathcal{G} \times \mathbb{Z}_{|\mathcal{G}|}$, where $\mathbb{G} = (\mathcal{G}, \cdot)$ is a finite, cyclic group with generator $g \in \mathcal{G}$. It might

```
spec $\hat{\mathbf{G}}$: // $\mathcal{X}$ points to $X_1$ (resp. $Y_1$); $\mathcal{A}$ to $A_2$
  var $D$ object; $d$ bool
  op (SETUP): $D$(SETUP); $d \leftarrow 0$
  op (1, WIN): ret ($d = 1$)
  op$^{\mathcal{X},\mathcal{A}}$ (1, OUT): $d \leftarrow D_1^{\mathcal{X},\mathcal{A}}$(OUT)
  op$^{\mathcal{X},\mathcal{A}}$ (2, $x$ any): ret $D_2^{\mathcal{X},\mathcal{A}}(x)$

spec $\mathbf{A}$: // $\mathcal{D}$ points to $\hat{G}_1$; $\mathcal{X}$ to $X_2$
  op$^{\mathcal{D},\mathcal{X}}$ (1, OUT): $\mathcal{D}$(OUT)
  op$^{\mathcal{D},\mathcal{X}}$ (2, $x$ any): ret $\mathcal{X}(x)$
```

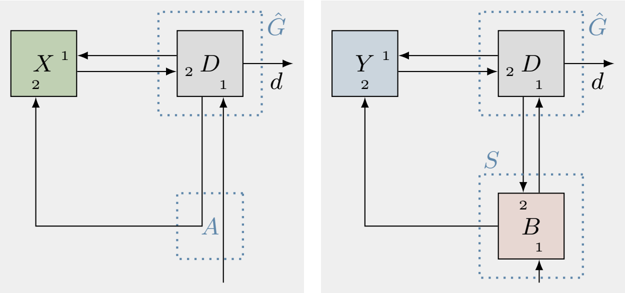

Figure 2.7: Left: Specs for Proof of Proposition 2. Right: Illustration of the reduction.

```
spec $\widetilde{\mathbf{Ex}}$: // $\mathcal{R}$ points to resources                spec Ex
 1  var $\Pi$ object, $PK$ elem$_\mathcal{G}$, $s$ int, $init$ bool
 2  op (SETUP): $\Pi$(SETUP); $init \leftarrow 0$
 3  // Main interface (intended usage)
 4  op$^{-,\mathcal{R}}$ (1, INIT):
 5      if $init = 1$ then ret $\perp$
 6      $init \leftarrow 1$; $s \twoheadleftarrow \mathbb{Z}_{|\mathcal{G}|}$; $PK \leftarrow g^s$; ret $PK$
 7  op$^{-,\mathcal{R}}$ (1, $x$ any):
 8      if $init = 1$ then ret $\Pi_s^{\mathcal{R}}(x)$
 9  // Auxiliary interface (additional usage)
10  op (2, PK): ret $PK$

11  op$^{-,\mathcal{R}}$ (2, SIGN, $msg$ str):
12      if $init = 0$ then ret $\perp$
13      $r \twoheadleftarrow \mathbb{Z}_{|\mathcal{G}|}$; $t \leftarrow \mathcal{R}_1(g^r, msg)$
14      ret $(r - st, t)$
```

```
spec Sim: // $\mathcal{X}$ points to $\tilde{X}_2$; $\mathcal{R}$ to resources
15  var $PK$ elem$_\mathcal{G}$
16  op$^{\mathcal{X},-}$ (2, PK): ret $\mathcal{X}$(PK)
17  op$^{\mathcal{X},\mathcal{R}}$ (2, SIGN, $msg$ str):
18      $PK \leftarrow \mathcal{X}$(PK)
19      if $PK = \perp$ then ret $\perp$
20      $M \leftarrow$ Src($PK, msg$)
21      $((\_, t), x) \leftarrow \mathcal{R}_1$(SET, $M$); ret $(x, t)$
22  op$^{-,\mathcal{R}}$ (3, ($R$ elem$_\mathcal{G}$, $msg$ str)):
23      ret $\mathcal{R}_1(R, msg)$

spec Src:
24  var $PK$ elem$_\mathcal{G}$, $msg$ str
25  op ( ):
26      $x, t \twoheadleftarrow \mathbb{Z}_{|\mathcal{G}|}$; $R \leftarrow g^x \cdot PK^t$
27      ret $(((R, msg), t), x)$
```

Figure 2.8: Left: Specifications **Ex** and $\widetilde{\mathbf{Ex}}$. Right: Specification of simulator S for the proof of Theorem 1.

be a public-key encryption (PKE) scheme, a digital signature (DS) algorithm, an authenticated key-exchange (AKE) protocol, etc. Now suppose the same key pair is deployed for an additional application, say, Schnorr's signature scheme (cf. Figure 2.2). We are interested in how this additional usage of the secret key impacts the existing security analysis for Π.[2]

System $\tilde{X} = \widetilde{\mathbf{Ex}}(\Pi)$ specified in Figure 2.8 captures the execution environment for Π in a security experiment. Its main interface lets the caller (i.e., the game) generate a key pair (PK, s) via the INIT-operator on lines 2.8:4-6 and make calls to $\Pi_s(\cdot)$ via the operator on lines 2.8:7-8. Its aux. interface provides the caller (i.e., the adversary) with access to PK (2.8:10). System $X = \mathbf{Ex}(\Pi)$ captures the same usage, except the adversary also gets a Schnorr-signing oracle (via the aux.-interface) for the secret key s (2.8:11-14). Presumably, the first resource in the experiment is instantiated with (a random oracle for) a hash function $H : \mathcal{G} \times \{0,1\}^* \to \mathbb{Z}_{|\mathcal{G}|}$.

Intuitively, the additional key usage captured by X should not significantly degrade the security of Π as long the adversary's interaction with the signing oracle cannot be used in an attack. We formalize this by exhibiting an efficient simulation of signing queries. When modeling H as an RO, a well-known strategy is to sample $x, t \twoheadleftarrow \mathbb{Z}_{|\mathcal{G}|}$ and "program" the RO so that $H(g^x \cdot PK^t, m) = t$, thereby ensuring that the arithmetic relationship between x and t is the same in the simulation as it is for the real signature. This strategy will fail if the programmed point overwrites a previous query, but the chance of this is small for reasonable parameters.

Theorem 1. *Suppose that Π is n.r. (not RO-programming). Then for every game G and t_A-time, (a_1, a_2, a_r)-query, n.d. adversary A there exists a $O((t_A)^2)$-time, $(a_1, a_2, a_2 + a_r)$-query, n.d. adversary B for which $\mathbf{Adv}^G_{X/H}(A) \leq \mathbf{Adv}^G_{\tilde{X}/\tilde{H}}(B) + 2a_2(a_2+q)/|\mathcal{G}|$, where H (resp. $\tilde{H}$) is an RO from $\mathcal{G} \times \{0,1\}^*$ to $\mathbb{Z}_{|\mathcal{G}|}$ with query limit $(q, 0)$ (resp. $(a_2 + q, 0)$).*

[2] This is a more general form of a question first posed by Haber and Pinkas [82], who formalized the security of PKE and DS schemes that share the same key pair.

Proof. Let $S = \mathbf{Sim}(\,)$ be as specified in Figure 2.8. This object simulates Schnorr signatures by calling $\mathcal{R}_1(\textsf{SET}, M)$, where source $M = \mathbf{Src}(PK, msg)$ is specified in Figure 2.8. This source generates x, t and computes the inputs to the RO just as described above and returns x as its hint. This causes the RO table T to be programmed so that $T[g^s \cdot PK^t, msg] = (x, t)$, where (x, t) is the signature returned by the simulator.

Let $N = |\mathcal{G}|$. First, observe that S is $(\log N, \log N)$-min-entropy, since each source it uses to program the RO chooses x and t uniformly and independently from $\mathbb{Z}_N$. The simulator is $(1, 1)$-query, since each of its operators makes at one $\mathcal{X}$-query (this points to $\tilde{X}_2$ in the reference experiment) and at most one $\mathcal{R}_1$-query (this points to H). Finally, its runtime is linear in the length of its inputs and in the computational cost of sampling elements of $\mathbb{Z}_N$ and performing the group operation in $\mathbb{G}$. Because the runtime of A includes the time required to evaluate its $\textsf{SIGN}$-queries, the runtime of S is $O(t_A)$.

Let P be an RO from $\mathcal{G} \times \{0,1\}^*$ to $\mathbb{Z}_N$ with query limit (q, a_2). Let $W = \mathbf{W}(G, X)$ and $\tilde{W} = \mathbf{Wo}(G, \tilde{X})$. By Lemma 1 there exist n.d. adversaries B and D' such that

$$\mathbf{Adv}^{G}_{X/H}(A) \leq a_2(a_2 + q)/N + \mathbf{Adv}^{G}_{\tilde{X}/\tilde{H}}(B) + \mathbf{Adv}^{\text{sr-indiff}^\psi}_{W/H, \tilde{W}/P}(D', S)\,, \tag{2.44}$$

where D' is $O(t_A)$-time and $(a_1{+}1, a_2, a_r)$-query, and B is $O((t_A)^2)$-time and $(a_1, a_2, a_2 + a_r)$-query. Suppose that G is (g_1, g_a)-query. Then by Lemma 2 there exists a $O(t_A)$-time, $((a_1{+}1)g_1, a_2, a_r)$-query, n.d. adversary D for which

$$\mathbf{Adv}^{\text{sr-indiff}^\psi}_{W/H, \tilde{W}/P}(D', S) \leq \mathbf{Adv}^{\text{sr-indiff}^\psi}_{X/H, \tilde{X}/P}(D, S)\,. \tag{2.45}$$

To complete the proof, we argue that

$$\mathbf{Adv}^{\text{sr-indiff}^\psi}_{X/H, \tilde{X}/P}(D, S) \leq a_2(a_2 + q)/N \tag{2.46}$$

for every such D. Observe that the simulation of the $\textsf{SIGN}$-operator provided by S is indistinguishable from the real operator as long as S never overwrites a point already set in the RO. Because each source specified by S is $(\log N, \log N)$-min-entropy and there are at most $a_2 + q$ distinct points in the RO table at any one time in the experiment, the probability that some point gets overwritten is at most $a_2(a_2 + q)/N$. ■

Remark 1. For the sake of presentation, the bound in Theorem 1 is expressed in terms of a query limit enforced by the RO. Normally we will allow the query limit to be unbounded so that the security bound is expressed in terms of the adversary's resources (e.g., Theorem 4). □

Chapter 3

Unspecified Behavior

As protocols like TLS have evolved, so have the formal tools used to analyze them. Often it is the protocol standards themselves, rather than implementations, that inspire and guide mathematical abstractions of these protocols; but their complexity makes the task of developing these abstractions quite challenging and prone to missing subtle attacks. Much of this complexity stems from the fact that protocols are only partially specified. The TLS 1.3 standard [123], whose record layer mechanism is the subject of this chapter, contains numerous "SHOULDs", "SHOULD NOTs" and "MAYs." Each of these provides a guideline, but not a rule (those are "MUSTs" and "MUST NOTs"), for compliant realizations of the standard. In this way, the TLS standard leaves many implementation details unspecified. This presents an important and interesting challenge for provable security: namely, deciding which of the standard's guidelines and unspecified behaviors are relevant to security, and so should be captured in the formal specification of the protocol.

The implications of these modeling choices are often clear only after an attack is found, leading to what Degabriele et al. call the "model-attack-remodel cycle" [64]. A prominent example is the case of padding-oracle attacks [146], which exploit weaknesses in the MAC-then-encrypt mode of operation used for authenticated encryption in many early secure channel protocols. This mode is provably secure [114], but only in a model in which decryption does not surface distinguishable errors. Yet compliant implementations of these protocols did make visible the cause of decryption failures (in particular, whether the encoding was invalid or the MAC was incorrect), leading to plaintext-recovery attacks [146, 66, 112]. The research community reacted by incorporating distinguishable errors into updated models [49, 75], but left more subtle attack vectors unaddressed [10], leading in turn to more sophisticated models [83, 18].

This reactive evolution of the security model is to be expected, but since standards only partially specify the protocol, it is hard to anticipate where vulnerabilities might arise in implementations. The Partially Specified Protocol (PSP) framework of Rogaway and Stegers [130] (introduced in Chapter 1) affords a more proactive approach that makes an explicit distinction in the security model between the parts of the standard that must be implemented correctly and those details which may be gotten wrong without impacting the security proof. In this chapter we provide a demonstration of this methodology in which we fully account for the unspecified behavior of the TLS 1.3 record layer [123].

Security Model for Secure Channels. Our formal model is described in §3.1. Its starting point is the *stream-based channel* abstraction introduced by Fischlin et al. [75] (hereafter FGMP). Their syntax accurately captures the APIs exposed by implementations of secure channels, in that it treats the sender- and receiver-side inputs and outputs as streams of message fragments, as opposed to atomic messages processed all at once. (It also admits distinguishable error messages.) We extend their syntax (cf. Def. 9) in order to account for multiplexing of many data streams over the same channel, as this is an essential feature of many secure channel protocols (including TLS).

We extend the FGMP notions of privacy and integrity to account for multiplexing. There are two main flavors of privacy (see Def. 10): the first, PRIV-S, is analogous to indistinguishability under chosen-plaintext attack, since the adversary only controls the sender's inputs; in the second, PRIV-SR, we also allow the adversary to mount chosen-ciphertext-fragment attacks. With each of these, we consider different "degrees" of privacy corresponding to various security goals considered in prior works [114, 75, 67]. We also consider integrity of the ciphertext stream written to the channel (INT-CS, Def. 11). Following FGMP, we show (in Lemma 3) how to achieve PRIV-SR security from a scheme that is both PRIV-S and INT-CS secure; just as with FGMP, we will need an additional property called *status simulatability* (SIM-STAT, Def. 12). Our notions are applicable to settings in which reliable transport (e.g., via TCP) is expected, and failure of the underlying transport mechanism to deliver stream fragments in order is deemed an attack (as in TLS and SSH[1] [103]).

Our Results. A number of behaviors not specified by the TLS 1.3 standard are relevant in the adversarial model of FGMP. For example, there are explicit rules that govern the manner in which plaintext fragments are buffered and coalesced into atomic plaintext records, but the specification leaves many design choices up to the implementation. We found the definitional viewpoint of the PSP methodology to be a useful tool for determining which pieces of the record layer specification are security critical and which are not. In particular, our analysis (Theorem 2, §3.2) uncovers two subtle and security-critical matters. First, the degree of privacy the record layer can provably provide depends intrinsically on unspecified behavior. The record layer is used to multiplex distinct plaintext streams over the same channel; thus, each record has a content type that associates the content to its stream. The content type is encrypted along with the content itself, permitting implementations that, at least in principle, hide both the content and its type. But the specification admits implementations that leak the content type entirely. Roughly speaking, this leakage occurs because the boundaries between records depend on the content type of each record. In general, we can conclude only that the record layer ensures privacy of the contents of each of the data streams.

Second, following FGMP, our notion of ciphertext-stream integrity implies that the receiver only consumes the stream produced by the sender. Records written to the channel are delimited by strings called *record headers*, whose values are specified by the standard. Our analysis applies to a draft of the unfinished standard (draft-ietf-tls-tls13-23 [124]) in which the header was not authenticated, and the standard did not require the receiver to check that their values for correctness. This early draft does not achieve our strong notion of ciphertext-stream integrity, although intuitively, the value of these bits should not impact security. Our framework provides a clean way to reconcile this intuition with our model: we show that the value of these bits are indeed irrelevant if and only if they are authenticated. (As we discuss in §3.2, this observation lead to a change in the final version of the standard, RFC 8446 [123].)

Limitations. Our analysis of TLS 1.3 demonstrates the effectiveness of the PSP methodology as a tool for taming the complexity of cryptographic standards—in particular, for dealing with unspecified behavior. Nevertheless, the TLS standard is subject to changes that our analysis would not account for. In §3.3 we discuss some limitations of our analysis and how the translation framework (presented in Chapter 2) could be used to overcome them.

Related Work. Our model for secure channels extends a long line of work: here we summarize some of the important landmarks in the development of the theory. In 2000, Bellare and Namprempre [26] provided foundations for the study of probabilistic authenticated encryption (AE) schemes used in SSL/TLS, IPSec [136], and SSH. Shortly thereafter, Rogaway [131] embellished authenticated encryption to take associated data (AEAD), moving the primitive closer to practice. Yet it was already understood that an AEAD scheme and its attendant notions of privacy and integrity do not suffice for building secure channels. In 2002, Bellare, Kohno, and Namprempre (BKN) [24] formalized stateful AE in order to account for replay and out-of-order delivery attacks, as well as to model and analyze SSH. Their model regards ciphertexts as atomic, but ciphertexts written to the channel may be (and routinely are) fragmented as they traverse the network, which leaves these protocols susceptible to attacks [9]. Likewise, the APIs for real secure channels regard the input plaintext as a stream, meaning that a single logical plaintext may be presented as a sequence of fragments, too. It took another ten years for the model to be significantly extended, by Boldyreva et al. [48], to address ciphertext fragmentation and attacks that exploit it. Finally, in 2015 by FGMP formalized stream-based secure channels that address plaintext fragmentation, with updates provided in 2016 by Albrecht et al. [8]. As FGMP point out [76], these works help shed formal light on truncation [140] and cookie-cutter [43] attacks.

From the standpoint of scope, the work most closely related to ours is Delignat-Lavaud et al. [67], who provide a mechanized proof of security for the TLS 1.3 record layer (draft-18). Our work is technically different from theirs on a couple fronts. First, our analysis applies to the set of compliant implementations (corresponding to different realizations of the specification details), whereas their work applies only to their implementation. Our notions are

[1] Secure SHell.

also more flexible: we capture the goal of hiding the message length as one of many possible privacy goals, whereas this property is mandatory in their security notion. Second, our adversarial model is stronger in that it permits fragmentation of the plaintext and ciphertext streams.

In an analysis of TLS 1.2, Paterson et al. [114] put forward a notion of stateful, length-hiding AE that admits schemes with associated padding (to hide the plaintext length) and variable-length MACs, both features of TLS 1.2. Their formalism necessarily elides a number of details of the protocol.

Badertscher et al. [13] characterize the TLS 1.3 record layer as an "augmented" secure channel (ASC), which allows for sending a message with two parts: the first being private, and both parts being authenticated. Their paper applies to draft-08, which mandated that the record sequence number, content type, and record version are all in the scope of the associated data for the AEAD computation. The content type was removed from the associated data in draft-09 and the sequence number and record version were removed in draft-11, leaving the associated data empty in draft-23 (the subject of our study).

Bellare and Tackmann analyze the multi-user security of the TLS 1.3 record layer [34], shedding light on the following question: if the same message is encrypted in a number of sessions, then what information does this leak about the sessions? A popular TLS endpoint might serve billions of clients a day. Many of these flows are identical (such as the initial GET); thus, an adversary who observes these identical flows can try to guess the key used for one of the clients. Its odds are improved by the sheer number of clients encrypting an identical message. This attack lead the designers of TLS 1.3 to "randomize" the IV used for generating the nonce: Bellare and Tackmann analyze the concrete security of this approach in the multi-user setting.

Preliminary Work. A preliminary version of the analysis in this chapter appears at CCS 2018 [116]. Here we have recast the security notions in the translation framework, simplified the specification of the record layer, and regenerated its proof of security.

3.1 Partially Specified Channels

Building on FGMP [75], our syntax models the computations of a sender and receiver communicating over an insecure (but reliable) channel. We assume the communicants are in possession of a shared secret, generated as specified by the scheme.

3.1.1 Syntax and Execution Environment

Definition 9 (Channels)**.** A *channel* is a halting object $\mathit{\Lambda}$ that exports the following operators:

– $(\textsf{GEN})\text{-}(K\ \textbf{any})$: generates a key K shared by the sender and receiver. This operator is stateless.

– $(K\ \textbf{any}, \textsf{MUX}, \mathit{msg}, \mathit{ctx}\ \textbf{str})\text{-}(x\ \textbf{str}, \alpha\ \textbf{any})$: inputs a stream fragment msg and stream context ctx and returns a pre-channel fragment x and auxiliary information α. We call this the *(stream-)multiplexing* operator.

– $(K\ \textbf{any}, \textsf{WRITE}, x\ \textbf{str}, \alpha\ \textbf{any})\text{-}(c\ \textbf{str}, \gamma\ \textbf{any})$: inputs a pre-channel fragment x and auxiliary information α and returns a channel fragment c and status information γ. We call this the *(channel-)writing* operator.

– $(K\ \textbf{any}, \textsf{READ}, c\ \textbf{str})\text{-}(y\ \textbf{str}, \alpha\ \textbf{any})$: inputs a channel fragment c and returns a channel fragment y and auxiliary information α. We call this the *(channel-)reading* operator.

– $(K\ \textbf{any}, \textsf{DEMUX}, y\ \textbf{str}, \alpha\ \textbf{any})\text{-}(\mathit{msg}, \mathit{ctx}\ \textbf{str}, \gamma\ \textbf{any})$: inputs a channel fragment y and auxiliary information α and returns a stream fragment msg, its context ctx, and status information γ. We call this *(stream-)demultiplexing* operator. □

Explanation. The input to the sender is a sequence of message fragments and the context of each fragment, the latter identifying the stream to which the fragment belongs. Here the interpretation of "identity" is up to the channel: for TLS, the context corresponds to fragment's content type (cf. §3.2.1). The stream multiplexer's task is to coalesce this sequence of stream fragments into a sequence of pre-channel fragments, which are consumed by the channel writer. The output of the channel writer is a sequence of channel fragments, which are consumed by the receiver.

```
spec Chan:  // $\mathcal{A}$ points to $A_2$ (via $G_2$); $\mathcal{R}$ to resources
 1  var $\Gamma, \mathit{Send}, \mathit{Recv}$ object, $K$ any
 2  op (SETUP): $\Gamma$(SETUP)
 3  op$^{\mathcal{A},\mathcal{R}}$ (1, INIT):
 4      $K \leftarrow \Gamma^{\mathcal{A},\mathcal{R}}$(GEN)
 5      $\mathit{Send} \leftarrow \mathit{Recv} \leftarrow \Gamma$
 6  op$^{-,\mathcal{R}}$ (1, RO, $x$ any): ret $\mathcal{R}(x)$
 7  op$^{\mathcal{A},\mathcal{R}}$ (1, SEND, $\mathit{msg}, \mathit{ctx}$ str):
 8      $(x, \alpha) \leftarrow \mathit{Send}_K^{\mathcal{A},\mathcal{R}}$(MUX, $\mathit{msg}, \mathit{ctx}$)
 9      $(c, \gamma) \leftarrow \mathit{Send}_K^{\mathcal{A},\mathcal{R}}$(WRITE, $x, \alpha$)
10      ret $(x, c, \gamma)$
11  op$^{\mathcal{A},\mathcal{R}}$ (1, RECV, $c$ str):
12      $(y, \alpha) \leftarrow \mathit{Recv}_K^{\mathcal{A},\mathcal{R}}$(READ, $c$)
13      $(\mathit{msg}, \mathit{ctx}, \gamma) \leftarrow \mathit{Recv}_K^{\mathcal{A},\mathcal{R}}$(DEMUX, $y, \alpha$)
14      ret $(y, \mathit{msg}, \mathit{ctx}, \gamma)$
```

```
spec Priv-S:  // $\mathcal{X}$ points to $X_1$; $\mathcal{A}$ to $A_2$                Priv-SR
15  var $\ell$ object, $b, w, \mathit{init}, \mathit{guess}, \mathit{sync}$ bool; $s$ str
16  op (SETUP): $b \twoheadleftarrow \{0, 1\}$
17      $w, \mathit{init}, \mathit{guess} \leftarrow 0$; $\mathit{sync} \leftarrow 1$; $s \leftarrow \varepsilon$
18  op (1, WIN): ret $w$
19  op (1, GUESS, $d$ bool):
20      if $\mathit{guess} \neq 1$ then $\mathit{guess} \leftarrow 1$; $w \leftarrow (b = d)$
21  op$^{\mathcal{X}}$ (1, INIT): if $\mathit{init} \neq 1$ then $\mathit{init} \leftarrow 1$; ret $\mathcal{X}$(INIT)
22  op$^{\mathcal{X}}$ (1, SEND, $\mathit{msg}_1, \mathit{ctx}_1, \mathit{msg}_0, \mathit{ctx}_0$ str):
23      if $\mathit{init} \neq 1$ then ret $(\bot, \bot)$
24      if $\ell(\mathit{msg}_1, \mathit{ctx}_1) \neq \ell(\mathit{msg}_0, \mathit{ctx}_0)$ then ret $(\bot, \bot)$
25      $(\_, c, \gamma) \leftarrow \mathcal{X}$(SEND, $\mathit{msg}_b, \mathit{ctx}_b$)
26      $s \leftarrow s \,\|\, c$; ret $(c, \gamma)$
    ┌──────────────────────────────────────────────
27  │ op$^{\mathcal{X}}$ (1, RECV, $c$ str):
28  │     if $\mathit{init} \neq 1$ then ret $(\bot, \bot, \bot)$
29  │     $(y, \mathit{msg}, \mathit{ctx}, \gamma) \leftarrow \mathcal{X}$(RECV, $c$)
30  │     if $\mathit{sync} = 1 \wedge y \preceq s$ then
31  │         $s \leftarrow s \,\%\, y$; ret $(\bot, \bot, \gamma)$
32  │     else $\mathit{sync} \leftarrow 0$; ret $(\mathit{msg}, \mathit{ctx}, \gamma)$
    └──────────────────────────────────────────────
33  op$^{-,\mathcal{A}}$ (2, $x$ any): ret $\mathcal{A}(x)$
```

```
spec Int-CS:
34  var $w, \mathit{init}, \mathit{sync}$ bool, $s$ str
35  op (SETUP): $w, \mathit{init} \leftarrow 0$; $\mathit{sync} \leftarrow 1$; $s \leftarrow \varepsilon$
36  op (1, WIN): ret $w$
37  op$^{\mathcal{X}}$ (1, INIT): if $\mathit{init} \neq 1$ then $\mathit{init} \leftarrow 1$; ret $\mathcal{X}$(INIT)
38  op$^{\mathcal{X}}$ (1, SEND, $\mathit{msg}, \mathit{ctx}$ str):
39      if $\mathit{init} \neq 1$ then ret $(\bot, \bot)$
40      $(\_, c, \gamma) \leftarrow \mathcal{X}$(SEND, $\mathit{msg}, \mathit{ctx}$)
41      $s \leftarrow s \,\|\, c$; ret $(c, \gamma)$
42  op$^{\mathcal{X}}$ (1, RECV, $c$ str):
43      if $\mathit{init} \neq 1$ then ret $(\bot, \bot, \bot)$
44      $(y, \mathit{msg}, \mathit{ctx}, \gamma) \leftarrow \mathcal{X}$(RECV, $c$)
45      if $\mathit{sync} = 1 \wedge y \preceq s$ then
46          $s \leftarrow s \,\%\, y$
47      else
48          $\mathit{sync} \leftarrow 0$
49          $w \leftarrow w \vee (\mathit{msg} \neq \bot \wedge \mathit{ctx} \neq \bot)$
50      ret $(\mathit{msg}, \mathit{ctx}, \gamma)$
51  op$^{-,\mathcal{A}}$ (2, $x$ any): ret $\mathcal{A}(x)$
```

Figure 3.1: Top: Execution environment **Chan**(Λ) for channel Λ. Bottom: Games **Priv-S**, **Priv-SR**, and **Int-CS** used to define various security properties for channels. Object ℓ is functional and halting.

The channel reader's task is to buffer channel fragments until one is ready to be processed: its output (also a channel fragment) is passed to the stream demultiplexer.

Execution of a Unidirectional Channel. System $X = \mathbf{Chan}(\Lambda)$ defined in Figure 3.1 formalizes the execution of a channel Λ in the security experiments defined in this chapter. Communication flows in one direction: from the sender, denoted by *Send*; to the receiver, denoted by *Recv*. The system exports four operators via its main interface (i.e., the interface called by the game in the security experiment). The first, INIT, generates a key according to Λ and stores it as K. The second, RO, provides the game with access to the shared resources defined externally in the encompassing experiment. (See "$\vec{R}$" in Figure 1.1.) The third operator, SEND (3.1:7-10), takes as input a stream fragment *msg* and its context *ctx* and executes the multiplexing and writing operators; the resulting channel fragment c and status info γ are returned as output. In addition, the operator returns the pre-channel fragment x output by the multiplexing operation. The last operator, RECV (3.1:11-14), takes as input a channel fragment c and executes the reading and demultiplexing operators; the resulting stream fragment *msg*, stream context *ctx*, and status info γ are returned as output. In addition, the operator returns the channel fragment y output by the reading operation.

Let us unpack this system a bit. The intermediate outputs x, y do not serve a functional purpose, but a technical one in definitions of security: in particular, the channel fragment y output by the SEND-operator will be used to determine if the channel is "in-sync" (cf. §3.1.2). The status information γ allows the sender and receiver to surface information for applications about the state of the channel. Making this information explicit in the syntax and execution environment allows us to model distinguishable decryption errors [49], an attack vector that has heavily influenced the development of secure channels [146, 66, 112, 10]. Our treatment is inspired by Barwell et al. [18], whose "subtle AE" setting models decryption leakage in a manner general enough to capture error indistinguishability [49, 75], as well as other settings for authenticated encryption [12, 83].

Remark 2. Absent from our syntax and execution semantics for channels is a notion of correctness. Indeed, because we allow channels to be partially specified (i.e., defined in terms of calls to the adversary's auxiliary interface), security is defined even for channels for which the adversary's SD-oracle responses may result in incorrect operation. One consequence of this modeling choice is that we cannot assume correctness in our security proofs, as do Bellare-Namprempre [26, Theorem 3.1] in the case of stateless and randomized AE, and FGMP [76, Appendix C] for stream-based channels. □

3.1.2 PRIV-SR and PRIV-S Security

We recast the privacy notions of FGMP to address multiplexing of plaintext data streams. Our PRIV-SR notion, specified by **Priv-SR** in Figure 3.1, gives the adversary access to a pair of operators. The first, SEND, allows the adversary to provide the sender with arbitrary stream fragment/context pairs. Analogously, the RECV-operator allows the adversary to deliver arbitrary channel fragments to the receiver. We define PRIV-S (**Priv-S**) the same way, except that the adversary does not have access to the RECV-operator. PRIV-S captures chosen plaintext attack security only, while PRIV-SR additionally captures a chosen ciphertext attack capability.

Channel Synchronization. Following prior work [24, 48, 75] we keep track of whether the channel is "in-sync" at any given moment during the adversary's attack. Loosely, the channel is said to be in-sync if the ciphertext "consumed" by the receiver, so far, is a prefix of the ciphertext output by the sender. In order to avoid trivial distinguishing attacks in the PRIV-SR game, it is necessary to suppress the message fragments output by the receiver while the channel is in-sync. We say the channel is in-sync as long as the sequence of channel fragments output by the channel reader is a prefix of the ciphertext stream output by the channel reader so far. Thus, the channel reader models receiver-side buffering and defragmentation of the channel.

Remark 3. This restricts the behavior of the receiver-side code in a way not seen in FGMP or its predecessors, but the restriction is relatively minor: a natural division of labor is to have the channel-reading operator buffer the ciphertext stream and output ciphertexts that are ready to decrypt; the job of stream demultiplexer, then, is to decrypt and process the output streams. This cleanly separates the tasks of "buffering" and "consuming" the ciphertext. The alternative would be to leave the receiver operations atomic, as FGMP have done. But this leads to much more complex security notions, as it requires handling synchronicity for a number of different cases (cf. [76, Def. 4.1]). □

Leakage Parameter. Our privacy notions are parameterized by an object ℓ, called the *leakage parameter*, that dictates the level of privacy required for security. As shown in Figure 3.1, the game's SEND-operator formalizes a left-or-right-style indistinguishability property: the adversary provides two stream fragment/context pairs to its oracle, only one of which is processed; and the pair that is processed depends on the outcome of a coin flip performed at the start of the experiment. A "valid" SEND-query $(msg_1, ctx_1, msg_0, ctx_0)$ is one for which $\ell(msg_1, ctx_1) = \ell(msg_0, ctx_0)$, i.e., the leakage of the left input is the same as the right. Intuitively, the less information that ℓ preserves about its inputs, the stronger the privacy requirement. For example, if ℓ computes the map $(msg, ctx) \mapsto (|msg|, ctx)$, then this permits channels that leak the length and context of each stream fragment. (As we will show in §3.2, this level of leakage is typical for implementations of secure channels.) On the other hand, the leakage parameter that computes the map $(msg, ctx) \mapsto \top$ hides both the length and context of each fragment, a level of security required in some security models [114, 67].

Refer to **Priv-SR** defined in Figure 3.1. Let Λ be a channel and let ℓ be a functional, halting object. Let $X = \mathbf{Chan}(\Lambda)$, $G = \mathbf{Priv\text{-}SR}(\ell)$, and $W = \mathbf{Wo}(G, X)$. Then world W specifies an adversary's capabilities in attacking the privacy of Λ with leakage parameter ℓ. The experiment begins by choosing a random bit b (via SETUP) and generating a secret key and distributing it to the sender and receiver (via a call to the INIT-operator of the main interface; see line 3.1:21). Thereafter, the adversary interacts with two operators.

The first, SEND (3.1:22-26), operates on quadruples of strings $(msg_1, ctx_1, msg_0, ctx_0)$. If the query is valid, then it computes $(\_, c, \gamma) \leftarrow \mathcal{X}(\text{SEND}, msg_b, ctx_b)$, where $\mathcal{X}$ points to X_1, and returns (c, γ) to the adversary. The channel fragment c is appended to a string s, which keeps track of the ciphertext stream produced by the sender but not yet consumed by the in-sync receiver.

```
spec $\mathbf{Sim\text{-}Stat}$:  // $\mathcal{X}$ points to $X_1$; $\mathcal{A}$ to $A_2$
var $S$ object; $b, w, init, guess$ bool, $s$ str
op (SETUP): $b \twoheadleftarrow \{0,1\}$
    $S$(SETUP); $w, init, guess \leftarrow 0$; $s \leftarrow \varepsilon$
op (1, WIN): ret $w$
op (1, GUESS, $d$ bool):
    if $guess \neq 1$ then $guess \leftarrow 1$; $w \leftarrow (b = d)$
op$^{\mathcal{X}}$ (1, INIT): if $init \neq 1$ then $init \leftarrow 1$; ret $\mathcal{X}$(INIT)
op$^{\mathcal{X}}$ (1, SEND, $msg, ctx$ str):
    if $init \neq 1$ then ret $(\bot, \bot)$
    $(\_, c, \gamma) \leftarrow \mathcal{X}(\text{SEND}, msg, ctx)$
    $s \leftarrow s \,\|\, c$; ret $(c, \gamma)$
op$^{\mathcal{X},\mathcal{A}}$ (1, RECV, $c$ str):
    var $\gamma$ any
    if $init \neq 1$ then ret $\bot$
    if $b = 1$ then
        $(\_, \_, \_, \gamma) \leftarrow \mathcal{X}(\text{RECV}, c)$
    else
        $\gamma \leftarrow S^{\mathcal{A}}(s, c)$
    ret $\gamma$
op$^{-,\mathcal{A}}$ (2, $x$ any): ret $\mathcal{A}(x)$
```

Figure 3.2: Specification **Sim-Stat** for defining SIM-STAT security.

The second operator, RECV (3.1:27-32), operates on channel fragments c. It first computes $(y, msg, ctx, \gamma) \leftarrow \mathcal{X}(\text{RECV}, c)$. Then, if the channel is in-sync (i.e., $sync = 1$) and the channel fragment y is a prefix of s ($y \preceq s$), then the first $|y|$ bits of s are removed from s and the operator returns $(\bot, \bot, \gamma)$ (3.1:31). This effectively suppresses the receiver's output while the channel is in-sync, thereby preventing the adversary from decrypting challenge ciphertexts. If the channel is out-of-sync or the channel is in-sync but y is not a prefix of s, then the channel is declared to be out-of-sync and the operator outputs (msg, ctx, γ), i.e., it does not suppress the output. Eventually the adversary guesses the value of b by making a query to the game's GUESS-operator.

Our model allows us to formalize channels that are only partially specified, as follows. Whenever a channel operation is executed, it is given oracle access to the adversary's auxiliary interface for making SD queries. These queries (made on lines 3.1:4, 8-9, and 12-13) are "routed" by the system to the game, which simply forwards them to the adversary (3.1:33).

Definition 10 (PRIV-SR$^\ell$ and PRIV-S$^\ell$ security)**.** Let Λ be channel and ℓ be a leakage parameter. Let $X = \mathbf{Chan}(\Lambda)$ and let $\vec{R}$ be resources. Define the PRIV-SR$^\ell$ advantage of adversary A in attacking $\Lambda/\vec{R}$ as

$$\mathbf{Adv}^{\text{priv-sr}^\ell}_{\Lambda/\vec{R}}(A) := 2\mathbf{Adv}^{\mathbf{Priv\text{-}SR}(\ell)}_{X/\vec{R}}(A) - 1\,.$$

Likewise, Define the PRIV-S$^\ell$ advantage of adversary A in attacking $\Lambda/\vec{R}$ as

$$\mathbf{Adv}^{\text{priv-s}^\ell}_{\Lambda/\vec{R}}(A) := 2\mathbf{Adv}^{\mathbf{Priv\text{-}S}(\ell)}_{X/\vec{R}}(A) - 1\,.$$

Informally, we say that $\Lambda/\vec{R}$ is PRIV-SR$^\ell$ secure (resp. PRIV-S$^\ell$ secure) if the advantage of every efficient adversary is small. □

3.1.3 INT-CS Security

Figure 3.1 defines another game **Int-CS** that captures the integrity of the ciphertext stream produced by the sender. Let Λ be a channel, let $X = \mathbf{Chan}(\Lambda)$, let $G = \mathbf{Int\text{-}CS}(\,)$, and let $W = \mathbf{Wo}(G, X)$. World W specifies an adversary's capabilities in attempting to break the ciphertext-stream integrity of Λ. Its goal is to fool the receiver into outputting a stream fragment while the channel is out-of-sync. The experiment begins by generating a key and distributing it to the sender and receiver. Thereafter, the adversary interacts with SEND- and RECV-operators defined in much the same way as in the privacy games, except that the SEND-operator only takes in a single stream fragment/context pair (msg, ctx). The RECV-operator sets a flag w if ever the channel is out-of-sync (i.e., $sync = 0$) and the receiver outputs a stream fragment/context pair.

Definition 11 (INT-CS security)**.** Let Λ be channel, let $X = \mathbf{Chan}(\Lambda)$, and let $\vec{R}$ be resources. Define the INT-CS advantage of adversary A in attacking $\Lambda/\vec{R}$ as $\mathbf{Adv}^{\text{int-cs}}_{\Lambda/\vec{R}}(A) := \mathbf{Adv}^{\mathbf{Int\text{-}CS}(\,)}_{X/\vec{R}}(A)$. Informally, we say that $\Lambda/\vec{R}$ is INT-CS secure if the advantage of every efficient adversary is small. □

3.1.4 SIM-STAT Security and a Generic Composition

If a channel is INT-CS secure, then an efficient attacker can do nothing but deliver the honestly produced ciphertext stream in the correct order. Thus, any channel that is both PRIV-S secure and INT-CS secure ought to be PRIV-SR secure as well, since the RECV-operator in the PRIV-SR experiment is useless to the attacker. This is almost true: the wrinkle is that the RECV-oracle returns status information in addition to the stream fragment and context. As in the FGMP setting, our syntax does not restrict the receiver (in particular, the demultiplexing operation) to return just one status message. Moreover, the status message may depend on the receiver state (of which a PRIV-S adversary would be ignorant), or be influenced by the adversarially controlled SD. In this section we give a notion for channels called SIM-STAT and show that it, PRIV-S, and INT-CS imply PRIV-SR.

The notion naturally captures what the adversary learns from the receiver's state by observing the status messages it outputs. It is inspired by ideas put forward in the subtle AE setting [18] and naturally generalizes a notion of FGMP. Let $\mathit{\Lambda}$ be a channel, let $X = \mathbf{Chan}(\mathit{\Lambda})$, let S be a simulator, let $G = \mathbf{Sim\text{-}Stat}(S)$ as specified in Figure 3.2, and let $W = \mathbf{Wo}(G, X)$. An adversary wins in world W if it correctly distinguishes the status information output by the receiver from the output of a simulator. The simulator has oracle access to the adversary's auxiliary interface (via $\mathcal{A}$) and gets as input the ciphertext stream s produced by the sender so far, as well as the incoming channel fragment c input to the receiver. These values allow the simulator to determine if the channel is in-sync.

Definition 12 (SIM-STAT security)**.** Let $\mathit{\Lambda}$ be a channel, S a simulator, and $\vec{R}$ be resources. Let $X = \mathbf{Chan}(\mathit{\Lambda})$. Define the SIM-STAT advantage of adversary A in attacking $\mathit{\Lambda}/\vec{R}$ relative to S as

$$\mathbf{Adv}^{\text{sim-stat}}_{\mathit{\Lambda}/\vec{R}}(A, S) := 2\mathbf{Adv}^{\mathbf{Sim\text{-}Stat}(S)}_{X/\vec{R}}(A) - 1\,.$$

Informally, we say that $\mathit{\Lambda}/\vec{R}$ is SIM-STAT secure if for every efficient adversary A there exists an efficient simulator S such that A's advantage in attacking $\mathit{\Lambda}/\vec{R}$ relative to S is small. □

Composition. The following lemma relates the PRIV-SR$^\ell$ security of a channel to its PRIV-S$^\ell$, INT-CS, and SIM-STAT security. The result is analogous to, but more general than, [76, Theorem 4.5]. It also confirms a conjecture of FGMP (cf. [76, Remark 4.6]).

Lemma 3. *Let $\mathit{\Lambda}$ be a channel, $\vec{R}$ be resources, and ℓ be a leakage parameter. For every t_A-time, n.d. adversary A making q_1 SEND-queries, q_2 RECV-queries, and q_r resource queries, and every t_S-time simulator S, there exist n.d. adversaries B, C, and D such that*

$$\mathbf{Adv}^{\text{priv-sr}^\ell}_{\mathit{\Lambda}/\vec{R}}(A) \le \mathbf{Adv}^{\text{priv-s}^\ell}_{\mathit{\Lambda}/\vec{R}}(B) + 2\mathbf{Adv}^{\text{int-cs}}_{\mathit{\Lambda}/\vec{R}}(C) + 2\mathbf{Adv}^{\text{sim-stat}}_{\mathit{\Lambda}/\vec{R}}(D, S)\,,$$

where B is $O(t_A + q_2 t_S)$-time and makes q_1 SEND-queries and q_r resource queries and C and D are $O(t_A)$-time and make q_1 SEND-, q_2 RECV-, and q_r resource queries.

Proof. At a high level, the idea is to construct a PRIV-S$^\ell$-adversary B from A, answering its RECV-queries using the simulator S. We will use INT-CS and SIM-STAT security to transition to an experiment in which this reduction can work. We proceed by a game-playing argument.

Experiment 0. We begin with a procedure $\mathbf{G}^0$ defined just like $\mathbf{Real}^{\mathsf{win}}_{W/\vec{R}}(A)$, where $W = \mathbf{Wo}(\mathbf{Priv\text{-}SR}(\ell), \mathbf{Chan}(\mathit{\Lambda}))$, except that the RECV-operator of **Priv-SR** is replaced with the code in the left panel of Figure 3.3. It is identical to the real experiment except that it sets flags w and bad if ever the channel is out-of-sync and the receiver outputs a stream fragment/context pair (msg, ctx) such that $msg \ne \bot \wedge ctx \ne \bot$. Spec **Priv-SR** is modified accordingly so that it declares variables bad, w **bool**. As this does not change the outcome of the experiment, we have that

$$\mathbf{Adv}^{\text{priv-sr}^\ell}_{\mathit{\Lambda}/\vec{R}}(A) = 2\Pr\left[\,\mathbf{G}^0(A)\,\right] - 1\,. \tag{3.1}$$

Revision 0-1. Define experiment $\mathbf{G}^1(A)$ from $\mathbf{G}^0(A)$ by adding the statement "$msg \leftarrow ctx \leftarrow \bot$" immediately after bad gets set. We can easily construct a $O(t_A)$-time, n.d. adversary C, with the same query complexity as A, that wins whenever bad gets set. Hence, by the fundamental lemma of game playing [33],

$$\Pr\left[\,\mathbf{G}^0(A)\,\right] \le \Pr\left[\,\mathbf{G}^1(A)\,\right] + \mathbf{Adv}^{\text{int-cs}}_{\mathit{\Lambda}/\vec{R}}(C)\,. \tag{3.2}$$

```
G^0
op^{X,A}(1, RECV, c str):
    if $init \neq 1$ then ret $(\bot, \bot, \bot)$
    $(y, msg, ctx, \gamma) \gets \mathcal{X}(\text{RECV}, c)$
    if $sync \wedge y \preceq s$ then
        $s \gets s \,\%\, y$; ret $(\bot, \bot, \gamma)$
    else
        $sync \gets 0$
        $w \gets w \vee (msg \neq \bot \wedge ctx \neq \bot)$
        if $w = 1$ then $bad \gets 1$
        ret $(msg, ctx, \gamma)$
```

```
G^1  [G^2]
op^{X,A}(1, RECV, c str):
    if $init \neq 1$ then ret $(\bot, \bot, \bot)$
    [ret $(\bot, \bot, S^{\mathcal{A}}(s, c))$]
    $(y, msg, ctx, \gamma) \gets \mathcal{X}(\text{RECV}, c)$
    if $sync \wedge y \preceq s$ then
        $s \gets s \,\%\, y$; ret $(\bot, \bot, \gamma)$
    else
        $sync \gets 0$
        $w \gets w \vee (msg \neq \bot \wedge ctx \neq \bot)$
        if $w = 1$ then $bad \gets 1$; $msg \gets ctx \gets \bot$
        ret $(msg, ctx, \gamma)$
```

Figure 3.3: Specifications for proof of Lemma 3.

Observe that, in the revised experiment, the RECV-operator outputs $msg = \bot$ and $ctx = \bot$ regardless of whether the channel is in-sync. This is precisely the behavior of the RECV-oracle in the SIM-STAT experiment when $b = 1$.

Revision 1-2. Define $\mathbf{G}^2(A, S)$ from $\mathbf{G}^1(A)$ by replacing the code for computing γ with execution of the simulator S, as shown in the right panel of Figure 3.3. This is precisely the behavior of the RECV-oracle in the SIM-STAT experiment when $b = 0$. There exists a $O(t_A)$-time, n.d. adversary D, with the same query complexity as A, for which

$$\Pr\left[\,\mathbf{G}^0(A)\,\right] \le \Pr\left[\,\mathbf{G}^2(A, S)\,\right] + \mathbf{Adv}^{\text{sim-stat}}_{\Lambda/\vec{R}}(D, S) + \mathbf{Adv}^{\text{int-cs}}_{\Lambda/\vec{R}}(C)\,. \tag{3.3}$$

Adversary C runs A, answering its queries using its own oracle interfaces. When A asks a query matching

$$(\text{SEND}, msg_1, ctx_1, msg_0, ctx_0\ \mathbf{str})$$

to its main interface, C first verifies that $\ell(msg_1, ctx_1) = \ell(msg_0, ctx_0)$, then returns $\mathcal{W}(\text{SEND}, msg_{d^*}, ctx_{d^*})$, where d^* is a random bit chosen by C during setup and $\mathcal{W}$ is the name of the main-interface given to A. When A asks $(\text{GUESS}, d\ \mathbf{bool})$, adversary C asks $\mathcal{W}(\text{GUESS}, (d = d^*))$ and halts. In other words, it guesses that its RECV-queries are being answered as usual (i.e., that $b = 1$ is its challenge bit) if A wins in its experiment; if A does not win, then C guesses that its queries are being answered by the simulator (i.e., that $b = 0$).

Experiment 2. We construct a $O(t_A + q_1 t_S)$-time, $(q_1, 0, q_r)$-query PRIV-S$^\ell$-adversary B for which the claim holds. On (SETUP), run $A(\text{SETUP})$; $S(\text{SETUP})$; and $s \gets \varepsilon$. On $(1, \text{WIN})$ with oracle interfaces $\mathcal{W}$, _, $\mathcal{R}$, return $A_1^{\mathbf{W},\_,\mathcal{R}}(\text{WIN})$. On input of $(2, x\ \mathbf{any})$ with oracle interface _, _, $\mathcal{R}$, return $A_2^{\_,\_,\mathcal{R}}(x)$. Adversary A's queries to $\mathbf{W}$ are answered as follows. On any input matching $(\text{SEND}, msg_1, ctx_1, msg_0, ctx_0\ \mathbf{str})$: run $(c, \gamma) \gets \mathcal{W}(\text{SEND}, msg_1, ctx_1, msg_0, ctx_0)$ and $s \gets s \,\|\, c$. Lastly, return (c, γ). On any input matching $(\text{RECV}, c\ \mathbf{str})$: return $(\bot, \bot, S^{\mathbf{A}}(s, c))$, where $\mathbf{A}(x) = A_2^{\mathcal{R}}(x)$. By construction, running B in the PRIV-S$^\ell$ experiment is equivalent to running $\mathbf{G}^2(A, S)$. We conclude that

$$\mathbf{Adv}^{\text{priv-s}^\ell}_{\Lambda/\vec{R}}(B) = 2\Pr\left[\,\mathbf{G}^2(A, S)\,\right] - 1\,. \tag{3.4}$$

Note that B's runtime is $O(t_A + q_2 t_S)$, since each of A's RECV-queries is answered by running S. ■

3.2 Case Study: TLS 1.3

Our study of partially specified channels owes much to a desire to analyze the TLS 1.3 record layer, in particular without eliding its optional features and unspecified behavior. So, we begin this section with an overview of some of its salient aspects, and a discussion of certain design choices that may have implications when the record layer is viewed through the lens of our security notions. This is followed (in §3.2.2) by a decomposition of the record layer into its component building blocks. Then we show how to securely compose these into a channel that faithfully captures the unspecified behavior of the TLS standard.

Note about the draft. This analysis pertains to draft-23 of the TLS 1.3 standard [124], which was current at the time of writing. Our analysis uncovers a potential attack vector in the draft: as a result, a change we suggested was adopted in the final version of the specification, RFC 8446 [123]. In the remainder, we distinguish draft-23 from the RFC by referring to the former as the "draft standard" and the latter as the "final standard".

3.2.1 Overview

TLS can be viewed as three protocols executing concurrently: the *handshake* protocol handles (re-)initialization of the channel; the *record* protocol is used to exchange application data between the client and the server; and the *alert* protocol is used to close the channel. The *record layer* refers to the mechanism used to protect flows between the client and server in each sub-protocol. Each flow is authenticated and encrypted as soon as the client and server have exchanged key material. (Usually the only unprotected messages are the initial "ClientHello" and "ServerHello" handshake messages.) Intuitively, each of these flows constitutes a logical data stream, and the record layer is a means of multiplexing these streams over a single communication channel. Among the record layer's many design criteria is the need to maximize flexibility for implementations. This means, somewhat paradoxically, that the specification does not fully specify every aspect of the scheme. Rather, the record layer specification (cf. [124, §5]) defines some core functionalities that must be implemented correctly and provides a set of guidelines and recommendations for compliant, fully realized schemes.

Content types. Each stream has an associated *content type*. Available types are handshake, application data, alert, and ChangeCipherSpec (CCS). Additional content types may be added, subject to certain guidelines (cf. [124, §11]). If the client or server receives a message of unknown content type, it must send an unexpected_message alert to its peer and terminate the connection. The CCS type is only available for compatibility with network infrastructure accustomed to processing records for TLS 1.2 and below. Usually a CCS message is treated as an unexpected message, but under specific conditions, it must be dropped.

Records. Plaintext records encode the content type, the stream fragment, the length of the fragment (which may not exceed 2^{14} bytes), and an additional field called *legacy_record_version*, whose value is fixed by the specification. (It is only present for backwards compatibility.) All flows, including unprotected ones (the Hellos and CCS messages) are formatted in this manner. The streams of data are transformed into a sequence of *records*. Stream fragments may be coalesced into a single record, but the *record boundaries* are subject to the following rules (cf. [124, §5.1]):

– (Handshake, no interleaving.) If two records correspond to a single handshake message, then they must be adjacent in the sequence of records.

– (Handshake, no spanning a key change.) If two records correspond to a single handshake message, then they both must precede the next key change (defined below). If this condition is violated, then the second record must be treated as an unexpected message.

– (Handshake and alert, no empty records.) Only application data records may have length 0.

– (One alert per record.) Alert messages must not be fragmented across records, and a record containing an alert message must contain only that message.

Additional content types must stipulate appropriate rules for record boundaries. Records are protected using an AEAD scheme (cf. [124, §5.2-5.4]). First, the record R is encoded as a string $X = R.\mathit{fragment} \parallel \mathit{str}_8(R.\mathit{type}) \parallel 0^{8p}$ for some $p \in \mathbb{N}$ such that the length of the ciphertext is less than $2^{14} + 256$ bytes. (Note the optional padding.) Function $\mathit{str}_m(\cdot)$ denotes a bijection from $\mathbb{Z}_m$ to $\{0,1\}^m$: for byte strings (i.e., when $m \equiv 0 \pmod 8$), this is the encoding of the unsigned integer input in big-endian byte-order.

In the draft standard, the padded record X is encrypted with associated data ε (i.e., the empty string) and a nonce N that we will define in a moment. The protected record has the following fields: integers *opaque_type*, *legacy_record_version*, and *length*; and string *encrypted_record*. The first, *opaque_type*, has a fixed value of 23. Similarly, *legacy_record_version* has a fixed value of 771 (or 0303 in hexadecimal). Field *length* indicates the length of *encrypted_record* in bytes.

The nonce N is computed from a sequence number *seqn* and an initialization vector IV (cf. [124, §5.3]). Both K and IV are derived from a shared secret (cf. [124, §7.1-§7.2]) using HKDF [92]. The length of IV is determined by the permitted nonce lengths of the AEAD scheme.[2] The nonce N is computed as $IV \oplus (0^{|IV|-64} \parallel str_{64}(seqn))$, where $0 \leq seqn \leq 2^{64} - 1$. Note that the client and server each uses a different key and IV for sending messages to the other. Thus, the record layer is executed as a unidirectional channel.

Usage Limits, Key Changes, and Protocol-Level Side-Effects. The spec mandates that the key be changed prior to the sequence number reaching its limit of $2^{64} - 1$ in order to prevent nonce reuse. It also recommends that implementations keep track of how many bytes of plaintext have been encrypted and decrypted with a single key and to change the key before the "safety limit" of the underlying AEAD scheme has been reached.

As mentioned above, upon receipt of a message of unknown type, the receiver should send its peer an unexpected_message alert. The alert stream is generally used to notify the recipient that the peer is tearing down its connection and will no longer write to the channel. There are *closure* alerts and *error* alerts [124, §6]. Both signal the tear down of the channel state, but they provide different feedback. The unexpected_message alert is an example of the latter. Error alerts are also used to indicate things like that the ciphertext is inauthentic or the record is malformed. An example of a closure alert is close_notify, which indicates that the receiver should not expect any more data from the peer, but that no error occurred.

The key and IV change during the normal course of the protocol. An update is always a side-effect of the handshake protocol: during transmission of application data, an update is signaled by a particular handshake message described in [124, §4.6.3], which informs the receiver that the sender has reinitialized its state and so must do so as well. The key change re-initializes the state of the sender and receiver with a fresh key and IV (derived from the shared secret), and the sequence number is set to 0 [124, §5.3]. Therefore, no sender or receiver state is held over after re-initialization of the channel.

Observations About the Standard. The standard defines some core functionalities, but leaves many design choices up to the implementer; our analysis aims to establish what security the record layer provides given this level of flexibility. Our approach is shaped by two questions. First, which fully specified components can be altered without impacting security? Second, which unspecified or partially specified behavior is security critical? We begin with a couple of observations.

The first is that the record boundaries may leak the content type (i.e., the stream context). The content type of each record is encrypted along with the fragment. The intent, presumably, is to hide both the content and its type, but the record boundary rules stipulated by the standard make hiding the type unachievable in general. Consider the one-alert-per-record rule, for example. The implementation is allowed to coalesce fragments of the same type, but a record containing an alert must contain only that alert. Thus, the length of each record output by the sender may, depending on the implementation, leak whether the record pertains to an alert or to application data. Of course, the standard does permit implementations that hide the content type of each record, but this is quite different from mandating this property. The takeaway is that encrypting the content type does not imply its secrecy, since the record boundaries depend on it.

One aspect of the scheme that is precisely defined is the format of the ciphertext transmitted on the wire. Each encrypted record begins with a header composed of *opaque_type*, *legacy_record_version*, and *length*. The values of the first two fields are fixed by the spec, and the last field is crucial for correct operation, since it informs the receiver of how many bytes to read next. What should the receiver do if the header is different than specified? Changing the *length* field bits should result in the next ciphertext either being too short or too long, and so would be deemed inauthentic with overwhelming probability. If *opaque_type* or *legacy_record_version* is mangled, then it should be safe to proceed since this does not affect the inputs to decryption.

However, doing so would be deemed an attack in our ciphertext-integrity (INT-CS) setting: changing these bits means the stream is out-of-sync, but since they are not authenticated (encryption uses ε for associated data), the receiver would successfully decrypt. In fact, checking the *opaque_type* and *legacy_record_version* fields is left optional by the spec: implementations MAY check these fields are correct and abort the connection if not (cf. [124, §5.2]). This presents us with a dilemma: if we leave this choice up to the specification details, then there is a trivial INT-CS attack. So, in order to salvage security, we need to lift this "MAY" to a "MUST".

This dilemma points to something rather strange about the draft standard's design: something that ought not be security critical—in particular, the value of the delimiter bits—is deemed an attack. Indeed, this observation

[2] The scheme must specify limits for valid nonce lengths, per RFC 5116 [106]. The maximum must be at least 8 bytes.

motivates our partially specified viewpoint. To formalize the idea that the value of the delimiter bits should not impact security, we simply let the specification details "choose" these bits itself. This is safe as long as the bits are authenticated and do not depend on sensitive values. (See §3.2.3 for details.)

An alternative conclusion is that this vulnerability is only an artifact of our adversarial model. Mangling the delimiter bits should not affect the inputs to decryption, and so does not constitute a "real attack" on privacy or integrity in an intuitive sense. To this point we offer a warning: this intuition is correct only if downstream handling of the plaintext is independent of the contents of these fields. Since such behavior is beyond the scope of the TLS standard (and even our security model), these legacy fields constitute an attack surface for implementations. The risk is not inconsiderable, as it is hard to predict how systems will evolve to make use of TLS, and of these bits in particular. Indeed, they owe their existence to backwards-compatibility requirements.

3.2.2 Building Blocks

In this section we formalize the core components of the record layer. Our aim is to sweep all but these essential building blocks into the specification details. The first primitive, called a stream multiplexer, captures the non-cryptographic functionality of the channel. It transforms the data streams into a sequence of pre-channel fragments (i.e., records), such that for each stream context (i.e., content type), the output of the receiver is a prefix of the input to the sender. TLS offers a great deal of flexibility with respect to the stream multiplexer's functionality: the flip side is that design choices here impact the security of the overall construction. (Recall the discussion on record boundaries in the previous section.) The second primitive is a scheme for authenticated encryption with associated data (AEAD).

Stream multiplexers. We define two security properties for stream multiplexers. Both are given below. (Explanation to follow.)

Definition 13 (Stream multiplexers)**.** A *stream multiplexer* is a halting object *Mux* that exports two operators:

- $(\textsf{MUX}, msg, ctx\ \textbf{str})$-$(X\ \textbf{str}, \gamma\ \textbf{any})$: inputs a stream fragment/context msg, ctx and outputs a pre-channel fragment X and status info γ.
- $(\textsf{DEMUX}, X\ \textbf{str})$-$(msg, ctx\ \textbf{str}, \gamma\ \textbf{any})$: inputs a pre-channel fragment X and returns a stream fragment/context msg, ctx and status info γ.

Refer to procedures $\mathbf{Exp}^{\text{mpriv-s}^\ell}_{b,Mux}(A)$ and $\mathbf{Exp}^{\text{sim-mstat}}_{b,Mux}(A)$ defined in Figure 3.4 and associated to a stream multiplexer *Mux*, adversary A, simulator S, leakage parameter ℓ, and bit b. Define the mPRIV-S$^\ell$ advantage of adversary A in attacking *Mux* as

$$\mathbf{Adv}^{\text{mpriv-s}^\ell}_{Mux}(A) := \Pr\big[\,\mathbf{Exp}^{\text{mpriv-s}^\ell}_{1,Mux}(A)\,\big] - \Pr\big[\,\mathbf{Exp}^{\text{mpriv-s}^\ell}_{0,Mux}(A)\,\big]\,.$$

Define the SIM-mSTAT advantage of adversary A in attacking *Mux* relative to S as

$$\mathbf{Adv}^{\text{sim-mstat}}_{Mux}(A,S) := \Pr\big[\,\mathbf{Exp}^{\text{sim-mstat}}_{1,Mux}(A,S)\,\big] - \Pr\big[\,\mathbf{Exp}^{\text{sim-mstat}}_{0,Mux}(A,S)\,\big]\,.$$

We say that *Mux* is mPRIV-S$^\ell$ (resp. SIM-mSTAT) secure if $\mathbf{Adv}^{\text{mpriv-s}^\ell}_{Mux}(A) = 0$ (resp. $\mathbf{Adv}^{\text{sim-mstat}}_{Mux}(A) = 0$) for all A, regardless of A's runtime or query complexity. (Note that security is defined information-theoretically, rather than computationally.) □

These security notions capture how design choices for this component of the channel impact the channel's overall security. The first, mPRIV-S$^\ell$, captures an adversary's ability to discern information about the stream fragment/context passed to $Mux(\textsf{MUX}, \cdot, \cdot)$ given (information about) its outputs. Like the PRIV-S$^\ell$ experiment (§3.1.2), the mPRIV-S$^\ell$ experiment has a leakage parameter ℓ. The adversary is given access to an oracle **Mux** with the same input pattern as the $\textsf{SEND}$-operator exported by the $\mathbf{Priv\text{-}S}(\ell)$ game (Figure 3.1), and its goal is to guess the value of the challenge bit b. (Here the challenge bit is written as a parameter of the experiment.) Where the experiments differ, however, is in the information available to the adversary. Rather than return the pre-channel fragment X directly, the oracle returns only the length of X (cf. 3.4:11). This captures a much weaker property than usual: rather than insisting that (X, γ) not leak anything about (msg, ctx) beyond $L = \ell(msg, ctx)$, we insist only that $(|X|, \gamma)$ not leak anything beyond L.

```
procedure $\mathbf{Exp}^{\mathrm{priv}}_{b,\mathit{AEAD}}(A)$i:
  $\mathcal{X}, \mathcal{Q} \gets \emptyset$; $A(\textsf{SETUP})$
  $K \twoheadleftarrow \mathcal{K}$; ret $A_1^{\mathbf{Enc}}(\textsf{OUT})$

procedure $\mathbf{Enc}(N, A, M)$:
  if $N \in \mathcal{X} \vee (N, A, M) \notin \mathcal{N} \times \mathcal{A} \times \mathcal{M}$ then ret $\bot$
  $\mathcal{X} \gets \mathcal{X} \cup \{N\}$; $c \gets \mathit{AEAD}(\textsf{CLEN}, |M|)$
  if $b = 1$ then ret $\mathit{AEAD}_K(\textsf{ENC}, N, A, M)$
  else $C \twoheadleftarrow \{0,1\}^c$; ret $C$
```

```
procedure $\mathbf{Exp}^{\mathrm{int}}_{\mathit{AEAD}}(A)$:
  $\mathcal{X}, \mathcal{Q} \gets \emptyset$; $A(\textsf{SETUP})$
  $K \twoheadleftarrow \mathcal{K}$; $w \gets 0$; $A_1^{\mathbf{Enc},\mathbf{Dec}}(\textsf{OUT})$; ret $w$

procedure $\mathbf{Enc}(N, A, M)$:
  if $N \in \mathcal{X} \vee (N, A, M) \notin \mathcal{N} \times \mathcal{A} \times \mathcal{M}$ then ret $\bot$
  $C \gets \mathit{AEAD}_K(\textsf{ENC}, N, A, M)$
  $\mathcal{X} \gets \mathcal{X} \cup \{N\}$; $\mathcal{Q} \gets \mathcal{Q} \cup \{(N, A, C)\}$; ret $C$

procedure $\mathbf{Dec}(N, A, C)$:
  $M \gets \mathit{AEAD}_K(\textsf{DEC}, N, A, C)$
  $w \gets w \vee [M \neq \bot \wedge (N, A, C) \notin \mathcal{Q}]$; ret $M$
```

```
procedure $\mathbf{Exp}^{\mathrm{mpriv\text{-}s}^{\ell}}_{b,\mathit{Mux}}(A)$:
  $\mathit{Mux}(\textsf{SETUP})$; $A(\textsf{SETUP})$
  ret $A_1^{\mathbf{Mux}}(\textsf{OUT})$

procedure $\mathbf{Mux}(\mathit{msg}_1, \mathit{ctx}_1, \mathit{msg}_0, \mathit{ctx}_0)$
  if $\ell(\mathit{msg}_1, \mathit{ctx}_1) \neq \ell(\mathit{msg}_0, \mathit{ctx}_0)$ then ret $(\bot, \bot)$
  $(X, \gamma) \gets \mathit{Mux}(\textsf{MUX}, \mathit{msg}_b, \mathit{ctx}_b)$
  ret $(|X|, \gamma)$
```

```
procedure $\mathbf{Exp}^{\mathrm{sim\text{-}mstat}}_{b,\mathit{Mux}}(A, S)$:
  $\mathit{Mux}(\textsf{SETUP})$: $A(\textsf{SETUP})$; $S(\textsf{SETUP})$
  ret $A_1^{\mathbf{Demux}}(\textsf{OUT})$

procedure $\mathbf{Demux}(X)$:
  if $b = 1$ then
      $(\_, \_, \gamma) \gets \mathit{Mux}(\textsf{DEMUX}, X)$
      ret $\gamma$
  else ret $S(|X|)$
```

Figure 3.4: Left: Experiments for defining PRIV (top) and INT (bottom) security of AEAD schemes. Sets $\mathcal{K}$, $\mathcal{N}$, $\mathcal{A}$, and $\mathcal{M}$ denote the key, nonce, AD, and message space of AEAD respectively. Right: Experiments for defining mPRIV-S$^{\ell}$ (top) and SIM-mSTAT (bottom) of stream multiplexers.

The second notion, SIM-mSTAT, captures simulatability of the status message output by $\mathit{Mux}(\textsf{DEMUX}, \cdot)$ given only the length of the pre-channel fragment. The adversary is given an oracle **Demux** that takes as input a pre-channel fragment X. If $b = 1$, then the oracle returns the status info about by $\mathit{Mux}(\textsf{DEMUX}, X)$; otherwise it executes the simulator S on input of $|X|$.

Authenticated Encryption with Associated Data. We describe the syntax for AEAD schemes as prescribed by the standard [106] and recall the standard notions of indistinguishability under chosen plaintext attack (simply called PRIV) and ciphertext integrity (INT).

Definition 14 (AEAD schemes)**.** An *AEAD scheme* is a halting, functional object AEAD that exports the following operators:

- $(K\ \mathbf{str}, \textsf{ENC}, N, A, M\ \mathbf{str})$-$(C\ \mathbf{str})$: encrypts message M under key K with nonce N and associated data (AD) A and returns the resulting ciphertext C.
- $(K\ \mathbf{str}, \textsf{DEC}, N, A, C\ \mathbf{str})$-$(M\ \mathbf{str})$: decrypts ciphertext C under key K with nonce N and AD A and returns the plaintext M.
- $(\textsf{CLEN}, m\ \mathbf{int})$-$(c\ \mathbf{int})$: returns the length c of the ciphertext corresponding to a plaintext of length m.
- $(\textsf{MLEN}, c\ \mathbf{int})$-$(m\ \mathbf{int})$: returns the length m of the plaintext corresponding to a ciphertext of length c.

We respectively define the key, nonce, associated data, and message space as the sets $\mathcal{K}, \mathcal{N}, \mathcal{A}, \mathcal{M} \subseteq \{0,1\}^*$ for which $\mathit{AEAD}_K(\textsf{ENC}, N, A, M) \neq \bot$ if and only if $(K, N, A, M) \in \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M}$. Correctness requires that $\mathit{AEAD}_K(\textsf{DEC}, N, A, \mathit{AEAD}_K(\textsf{ENC}, N, A, M)) = M$ for all such (K, N, A, M). (This condition implies that AEAD is both *correct* and *tidy* in the sense of Namprempre et al. [108].) We require that $|C| = \mathit{AEAD}(\textsf{CLEN}, |M|)$ and $|M| = \mathit{AEAD}(\textsf{MLEN}, |C|)$ for all such (K, N, A, M), where $C = \mathit{AEAD}_K(\textsf{ENC}, N, A, M)$. For all $M \in \{0,1\}^*$ we require that, if $M \in \mathcal{M}$, then $\{0,1\}^{|M|} \subseteq \mathcal{M}$. Finally, we require the sets $\mathcal{K}$, $\mathcal{N}$, $\mathcal{A}$, and $\mathcal{M}$ are computable.

Refer to procedures $\mathbf{Exp}^{\mathrm{priv}}_{b,\mathit{AEAD}}(A)$ and $\mathbf{Exp}^{\mathrm{int}}_{\mathit{AEAD}}(A)$ defined in Figure 3.4 and associated to a scheme AEAD, adversary A, and bit b. Define the PRIV advantage of A as

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathit{AEAD}}(A) := \Pr\big[\,\mathbf{Exp}^{\mathrm{priv}}_{1,\mathit{AEAD}}(A)\,\big] - \Pr\big[\,\mathbf{Exp}^{\mathrm{priv}}_{0,\mathit{AEAD}}(A)\,\big]$$

```
spec $\textbf{TLS-Record}_n$:  // $\mathcal{A}$ points to $A_2$
 1  var $AEAD, Mux$ object
 2  var $seqn$ int, $buf$ str, $sync$ bool
 3  op (SETUP): $Mux$(SETUP)
 4      $seqn \leftarrow 0$; $buf \leftarrow \varepsilon$; $sync \leftarrow 1$
 5  op (GEN):
 6      $(K, IV) \twoheadleftarrow \mathcal{K} \times \{0,1\}^n$; ret $(K, IV)$
 7
 8  op (MUX, $(K, IV$ str$)$, $msg$, $ctx$ str):
 9      $(X, \alpha) \leftarrow Mux$(MUX, $msg, ctx$)
10      if $X = \varepsilon$ then ret $(\varepsilon, (\varepsilon, \alpha))$
11      $N \leftarrow IV \oplus (0^{n-64} \,\|\, str_{64}(seqn))$
12      $seqn \leftarrow seqn + 1$
13      ret $(X, (N, \alpha))$
14
15  op$^{\mathcal{A}}$ (WRITE, $(K, IV$ str$)$, $X$ str, $(N$ str, $\alpha$ any$)$):
16      var $A$ str, $\gamma$ any
17      $(A, \gamma) \leftarrow \mathcal{A}$(send ready, $|X|, \alpha$)
18      if $X = \varepsilon$ then ret $(\varepsilon, \gamma)$
19      $Y' \leftarrow AEAD_K$(ENC, $N, A, X$)
20      if $Y' = \bot \vee \mathcal{A}$(parse, $A \,\|\, Y') \neq (|A| + |Y'|, |A|)$
21          then ret $(\varepsilon, \mathcal{A}$(invalid ptxt)$)$
22      ret $(A \,\|\, Y', \gamma)$

23  op$^{\mathcal{A}}$ (READ, $(K, IV$ str$)$, $C$ str):
24      var $u$ int, $drop$ bool, $\alpha$ any
25      $buf \leftarrow buf \,\|\, C$
26      $(u, \_) \leftarrow \mathcal{A}$(parse, $buf$); $(drop, \alpha) \leftarrow \mathcal{A}$(drop, $buf$)
27      $Y \leftarrow buf[:u]$; $buf \leftarrow buf \,\%\, Y$
28      if $Y = \varepsilon \vee drop = 1$ then ret $(\varepsilon, (\varepsilon, \alpha))$
29      $N \leftarrow IV \oplus (0^{n-64} \,\|\, str_{64}(seqn))$
30      $seqn \leftarrow seqn + 1$
31      ret $(Y, (N, \alpha))$
32
33  op$^{\mathcal{A}}$ (DEMUX, $(K, IV$ str$)$, $Y$ str, $(N$ str, $\alpha$ any$)$):
34      var $X$ str, $v$ int, $\gamma$ any
35      $X \leftarrow \varepsilon$; $\gamma \leftarrow \mathcal{A}$(recv ready, $|Y|, \alpha$)
36      if $(Y = \varepsilon \wedge \gamma \neq \bot) \vee sync = 0$ then ret $(\bot, \bot, \gamma)$
37      else if $Y \neq \varepsilon$ then
38          $(\_, v) \leftarrow \mathcal{A}$(parse, $Y$); $A \leftarrow Y[:v]$; $Y' \leftarrow Y \,\%\, A$
39          $X \leftarrow AEAD_K$(DEC, $N, A, Y'$)
40          if $X = \bot$ then
41              $sync \leftarrow 0$
42              ret $(\bot, \bot, \mathcal{A}$(invalid ctxt)$)$
43      $(msg, ctx, \gamma) \leftarrow Mux$(DEMUX, $X$)
44      ret $(msg, ctx, \gamma)$
```

Figure 3.5: Spec $\textbf{TLS-Record}_n$, a partial specification of the TLS 1.3 record layer. It is defined in terms of a stream multiplexer (Def. 13) and an AEAD scheme (Def. 14). Set $\mathcal{K}$ denotes the key space of $AEAD$; it must be finite. Integer $n \geq 64$ is the IV length; it is chosen so that $\{0,1\}^n \subseteq \mathcal{N}$, where $\mathcal{N}$ denotes the nonce space of $AEAD$. Function $str_m(\cdot)$ is a bijection from $\mathbb{Z}_m$ to $\{0,1\}^m$.

and the INT advantage of A as

$$\mathbf{Adv}^{\mathrm{int}}_{AEAD}(A) := \Pr\big[\,\mathbf{Exp}^{\mathrm{int}}_{AEAD}(A)\,\big]\,.$$

Informally, $AEAD$ is PRIV (resp. INT) secure if the PRIV (resp. INT) advantage of every reasonably efficient adversary is small. □

3.2.3 Security of the Record Layer

We are now ready to formalize the TLS 1.3 record layer as a partially specified channel. Let Mux be a stream multiplexer and let $AEAD$ be an AEAD scheme with a finite key space. Let $n \geq 64$ be an integer for which $\{0,1\}^n$ is a subset of the nonce space of $AEAD$. Define channel $\Lambda = \textbf{TLS-Record}_n(AEAD, Mux)$ as specified in Figure 3.5.

Before diving in, we first note that our formal specification of Λ differs from the draft standard in one small, but security-critical way. Namely, draft-23 mandates that the AEAD scheme be invoked with the empty string as the AD, whereas in our scheme, the string A (i.e., the record header) is used as AD. To fully comply with the spec, one would replace A with ε on lines 3.5:19 and 39. This change would lead to a trivial INT-CS attack. Suppose the sender outputs $Y = A \,\|\, Y'$. Then the adversary can deliver $A^* \,\|\, Y'$ to the receiver for some $A^* \neq A$ where $|A^*| = |A|$. If Y is consumed by the receiver, then the channel will be deemed out-of-sync by the INT-CS experiment, but not by the channel. This is inevitable in the stream-based channel setting: if one were to directly extend FGMP's syntax and security notions and analyze the record layer, then one would still encounter the same problem. In light of this limitation, this change was adopted in the final standard.

The Sender. The MUX-operator (3.5:8-13) invokes the stream multiplexer Mux in order to buffer the input stream and determine the next record (i.e., pre-channel fragment) to output. The channel is designed to never operate on 0-length records (3.5:10): if the pre-channel fragment X input to the channel writer (i.e., the WRITE-operator) is ε, then the channel writer outputs a 0-length channel fragment (3.5:18). Otherwise the record is protected and written to the channel. The data on the wire is $A \,\|\, Y'$, where Y' is the encryption of X and A is a string chosen by the SD (via a call to $\mathcal{A}$; cf. 3.5:17). We refer to A as the record header, Y' as the record data, and $A \,\|\, Y'$ as the record.

The Receiver. Defragmentation of records written to the channel is performed by the channel reader (i.e., the READ-operator on lines 3.5:23-31). First, the channel fragment C is appended to an internal buffer buf **str**. Next, the SD oracle ($\mathcal{A}$) is invoked in order to decide how much of the buffer to dequeue next. This is accomplished by a call of the form $\mathcal{A}(\text{parse}, buf)$. The value returned by this call is interpreted as a pair of integers (u, v), the first being the length of the record to consume next, and the second being the length of the record header. The channel reader dequeues the first u bits of buf corresponding to a complete record Y. The SD is then called upon to decide whether to drop Y without further processing (3.5:28). (This permits the rules for handling CCS messages.) If not, then presumably the channel fragment Y is the concatenation of a record header A, where $|A| = v$, and the corresponding record data Y'. If $Y \neq \varepsilon$, then Y' is decrypted (using A as associated data) and the resulting pre-channel fragment X (i.e., unprotected record) is input to the stream multiplexer Mux.

Once the DEMUX-operator (3.5:33-44) encounters an invalid ciphertext, it marks the channel as out-of-sync and never outputs another a fragment. If the receiver is in-sync and the channel fragment Y is 0-length, then the operator may poll Mux to see if a message fragment is available for outputting. (That is, Mux may be invoked on line 3.5:43 with $X = \varepsilon$.) Finally, usage limits are enforced by the SD, which may shutdown the channel by outputting an error as shown on lines 3.5:18 and 36.

Security. Fix $AEAD$, Mux, n, and $\Lambda = \textbf{TLS-Record}_n(AEAD, Mux)$ as specified above. Let ℓ be a leakage parameter. The following theorem says that the PRIV-SR$^\ell$ security of Λ follows from the PRIV and INT security of $AEAD$ and the mPRIV-S$^\ell$ and SIM-mSTAT security of Mux.

Theorem 2. *Let $0 \leq q_s, q_r < 2^{64}$ be integers. Let $A = \textbf{NoDeg}(M, SD)$ be a t_A-time, n.d., adversary that makes q_s SEND-queries and q_r RECV-queries. Suppose SD exports a $(\text{parse}, \_\ \textbf{str})$-$(\textbf{int}, \textbf{int})$-operator for which $SD(\text{parse}, Y) = SD(\text{parse}, buf)$ for all strings $Y \preceq buf$. Then for every t_S-time simulator S there exist adversaries B, C, D, and E such that*

$$\mathbf{Adv}_{\Lambda}^{\text{priv-sr}^\ell}(A) \leq \mathbf{Adv}_{Mux}^{\text{mpriv-s}^\ell}(B) + 2\mathbf{Adv}_{Mux}^{\text{sim-mstat}}(C, S) + 8\mathbf{Adv}_{AEAD}^{\text{int}}(D) + 2\mathbf{Adv}_{AEAD}^{\text{priv}}(E)\,,$$

*where: B is $O(q_r(t_S + t_A))$-time and makes at most q_s **Mux**-queries; C is $O(t_A)$-time and makes at most q_r **Demux**-queries; D is $O(t_A)$-time and makes at most q_s **Enc**-queries and q_r **Dec**-queries; and E is $O(q_r(t_S + t_A))$-time and makes at most q_s **Enc**-queries.*

Let us briefly address the restrictions on A before we proceed with the proof. Recall that the parse-operator exported by the adversary's auxiliary interface takes as input a channel fragment and returns the length u of the next complete record, and the length v of the corresponding record header. This information is used by the receiver (lines 3.5:26 and 38) to process the record, and it used by the sender (line 3.5:20) to ensure that each record it outputs can be correctly parsed by the receiver. This, coupled with the requirement that the adversary is non-degenerate (i.e., $\mathcal{A}$-queries are answered deterministically and statelessly; see Def. 4), ensures that record boundaries are determined solely by the sequence of channel fragments delivered to the receiver. This ought to be true of any correct implementation of the record layer; interestingly, it also turns out to be necessary in order to prove SIM-STAT security of Λ, which we use to derive the bound (via Lemma 3).

Proof of Theorem 2. By Lemma 3, for every $t_{S'}$-time simulator S' there exist n.d. adversaries B', C', and D' such that

$$\mathbf{Adv}_{\Lambda}^{\text{priv-sr}^\ell}(A) \leq \mathbf{Adv}_{\Lambda}^{\text{priv-s}^\ell}(B') + 2\mathbf{Adv}_{\Lambda}^{\text{int-cs}}(D') + 2\mathbf{Adv}_{\Lambda}^{\text{sim-stat}}(C', S')\,, \tag{3.5}$$

where B' is $O(t_A + q_r t_{S'})$-time and makes q_s SEND-queries, and C' and D' are both $O(t_A)$-time and each makes q_s SEND-queries and q_r RECV-queries. We proceed by bounding each term on the right hand side of Eq. (3.5). First, the PRIV-S$^\ell$ security of Λ follows from the mPRIV-S$^\ell$ security of Mux and the PRIV security of $AEAD$.

Claim 2. There exist $O(t_A + q_r t_{S'})$-time adversaries B and E for which

$$\mathbf{Adv}_{\Lambda}^{\text{priv-s}^\ell}(B') \leq \mathbf{Adv}_{Mux}^{\text{mpriv-s}^\ell}(B) + 2\mathbf{Adv}_{AEAD}^{\text{priv}}(E)\,,$$

where B makes q_s **Mux**-queries and E makes at most q_s **Enc**-queries.

```
procedure $\mathbf{SimSend}(msg_1, ctx_1, msg_0, ctx_0)$:
  var $Y', N, A$ str, $x$ int, $\alpha, \gamma$ any
  if $init \neq 1 \vee \ell(msg_1, ctx_1) \neq \ell(msg_0, ctx_0)$ then ret $(\bot, \bot)$
  $(x, \alpha) \leftarrow \mathbf{Mux}(msg_1, ctx_1, msg_0, ctx_0)$
  if $x = \varepsilon$ then $N \leftarrow \varepsilon$
  else $N \leftarrow IV \oplus (0^{n-64} \,\|\, str_{64}(seqn))$; $seqn \leftarrow seqn + 1$
  $(A, \gamma) \leftarrow \mathcal{A}(\textsf{send ready}, x, \alpha)$
  if $x = 0$ then ret $(\varepsilon, \gamma)$
  if $(N, A, 1^x) \notin \mathcal{N} \times \mathcal{A} \times \mathcal{M}$ then $Y' \leftarrow \bot$
  else $c \leftarrow AEAD(\textsf{CLEN}, x)$; $Y' \twoheadleftarrow \{0,1\}^c$
  if $Y' = \bot \vee \mathcal{A}(\textsf{parse}, A \,\|\, Y') \neq (|A| + |Y'|, |A|)$ then
    ret $(\varepsilon, \mathcal{A}(\textsf{invalid ptxt}))$
  ret $(A \,\|\, Y', \gamma)$
```

Figure 3.6: Procedure **SimSend** for the proof of Claim 2.

Proof. Let $\mathcal{N}$, $\mathcal{A}$, and $\mathcal{M}$ denote the nonce, AD, and message space of $AEAD$ respectively. Define procedure $\mathbf{G}^0$ so that $\mathbf{Adv}^{\text{priv-s}^\ell}_{\Lambda}(B') = 2\Pr\big[\,\mathbf{G}^0(B')\,\big] - 1$, i.e., $\mathbf{G}^0$ is defined to be the PRIV-S$^\ell$ experiment for Λ (see Def. 10). Define procedure $\mathbf{G}^1$ from $\mathbf{G}^0$ by modifying the specification of Λ so that "$Y' \leftarrow AEAD_K(\textsf{ENC}, N, A, X)$" on line 3.5:19 is replaced with the following code:

```
  var $Y'$ str
  if $(N, A, X) \notin \mathcal{N} \times \mathcal{A} \times \mathcal{M}$ then $Y' \leftarrow \bot$
  else $c \leftarrow AEAD(\textsf{CLEN}, |X|)$; $Y' \twoheadleftarrow \{0,1\}^c$
```

We exhibit a $O(t_{B'})$-time adversary E that makes at most q_s **Enc**-queries and for which

$$\Pr\big[\,\mathbf{G}^0(B')\,\big] \leq \Pr\big[\,\mathbf{G}^1(B')\,\big] + \mathbf{Adv}^{\text{priv}}_{AEAD}(E)\,. \tag{3.6}$$

Adversary E simply runs $\mathbf{G}^0(B')$ with line 3.5:19 replaced with "$Y' \leftarrow \mathbf{Enc}(N, A, X)$". Requiring that $q_s < 2^{64}$ ensures that the nonce of each **Enc**-query is unique, and so we have that $\Pr\big[\,\mathbf{Exp}^{\text{int}}_{1,AEAD}(E)\,\big] = \Pr\big[\,\mathbf{G}^0(B')\,\big]$ and $\Pr\big[\,\mathbf{Exp}^{\text{int}}_{0,AEAD}(E)\,\big] = \Pr\big[\,\mathbf{G}^1(B')\,\big]$.

Now consider the mPRIV-S$^\ell$ adversary B that runs $\mathbf{G}^1(B')$, except that it answers main-interface queries matching $(\textsf{SEND}, msg_1, ctx_1, msg_0, ctx_0 \ \mathbf{str})$ using the procedure in Figure 3.6 (cf. 3.1:22-26). This procedure begins as usual by checking that the game is initiated and that the inputs (msg_1, ctx_1) and (msg_0, ctx_0) have the same leakage. The adversary then invokes its own **Mux**-oracle on input of $(msg_1, ctx_1, msg_0, ctx_0)$, getting the length of the resulting pre-channel fragment in return. The rest of the procedure is just like $\mathbf{G}^1$, except that it operates on the length of the pre-channel fragment rather than the fragment itself. This does not change the output of the $\textsf{SEND}$-operator, however, since by the correctness of $AEAD$, $X \in \mathcal{M}$ implies that $1^{|X|} \in \mathcal{M}$ for all X.

When B' makes a main-interface query matching $(\textsf{GUESS}, d\ \mathbf{bool})$ (cf. 3.1:19), adversary B immediately halts and outputs d as its guess of the challenge bit in its own experiment. By construction we have that

$$\Pr\big[\,\mathbf{G}^1(B')\,\big] = \Pr\big[\,b \twoheadleftarrow \{0,1\};\, b' \twoheadleftarrow \mathbf{Exp}^{\text{mpriv-s}^\ell}_{b,Mux}(B) : b' = b\,\big]\,. \tag{3.7}$$

The claimed bound follows by conditioning on the outcome of b. Note that B is $O(t_{B'})$-time and makes at most q_s **Mux**-queries. □

Claim 3. There exists a $O(t_A)$-time adversary D_0 for which $\mathbf{Adv}^{\text{int-cs}}_{\Lambda}(D') \leq \mathbf{Adv}^{\text{int}}_{AEAD}(D_0)$ and D_0 makes at most q_s **Enc**-queries and q_r **Dec**-queries.

Proof. Let $W = \mathbf{World}(AEAD, Mux)$ be the world specified in Figure 3.7. It is functionally equivalent to the world prescribed by the INT-CS experiment for Λ: in particular, the probability that $\mathbf{Real}^{\mathsf{win}}_{W}(D') = 1$ is exactly the probability that $\mathbf{Real}^{\mathsf{win}}_{\hat{W}}(D') = 1$, where $\hat{W} = \mathbf{Wo}(\mathbf{Int\text{-}CS}(\,), \mathbf{Chan}(\Lambda))$. World W was obtained from the specification of **Int-CS** by replacing $\mathcal{X}$-queries with the code that is executed by the system in response to these queries. The sender's operations are implemented by procedure **Send**, defined in Figure 3.8, and the receiver's

```
spec $\mathbf{World}$:
 1  var $AEAD, Mux, M_1, M_0$ $\mathbf{object}$
 2  var $w, bad, sync, \widehat{sync}$ $\mathbf{bool}$, $K, IV, s, \hat{s}, \hat{t}$ $\mathbf{str}$
 3  var $T$ $\mathbf{table}$, $buf$ $\mathbf{str}$, $ct_1, ct_0$ $\mathbf{int}$
 4
 5  op (SETUP):
 6      $Mux$(SETUP); $M_1 \leftarrow M_0 \leftarrow Mux$
 7      $w, init, bad \leftarrow 0$; $sync, \widehat{sync} \leftarrow 1$; $T \leftarrow [\,]$
 8      $s, \hat{s}, \hat{t}, buf \leftarrow \varepsilon$; $ct_1, ct_0 \leftarrow 0$
 9
10  op (1, WIN): ret $w$
11
12  op (1, INIT):
13      if $init \neq 1$ then $init \leftarrow 1$
14      $(K, IV) \twoheadleftarrow \mathcal{K} \times \{0,1\}^n$
15  op$^{-,\mathcal{A}}$ (1, SEND, $msg$, $ctx$ $\mathbf{str}$):
16      if $init \neq 1$ then ret $(\bot, \bot)$
17      $(\_, c, \gamma) \leftarrow \mathbf{Send}(msg, ctx)$
18      $s \leftarrow s \,\|\, c$; $\hat{s} \leftarrow \hat{s} \,\|\, c$
19      ret $(c, \gamma)$
20
21  op$^{-,\mathcal{A}}$ (1, RECV, $c$ $\mathbf{str}$):
22      if $init \neq 1$ then ret $(\bot, \bot, \bot)$
23      $(Y, msg, ctx, \gamma) \leftarrow \mathbf{Send}(c)$; $\hat{t} \leftarrow \hat{t} \,\|\, Y$
24      if $sync = 1 \wedge Y \preceq s$ then
25          $s \leftarrow s \,\%\, Y$
26      else
27          $sync \leftarrow 0$
28          $w \leftarrow w \vee (msg \neq \bot \wedge ctx \neq \bot)$
29      ret $(msg, ctx, \gamma)$
```

Figure 3.7: Specification of world $W = \mathbf{World}(AEAD, Mux)$ for the proof of Claim 3. Procedures **Send** and **Recv** are defined in Figure 3.8.

operations are implemented by procedure **Recv**, also defined in Figure 3.8. The sender and receiver share access to the secret key K and IV IV, but each has its own copy of the stream multiplexer (M_1 for the sender and M_0 for the receiver) and maintains its own sequence number (ct_1 for the sender and ct_0 for the receiver). The flag used by the receiver to keep track of the channel's state has been renamed to $\widehat{sync}$. Finally, the sender and receiver share access to a table T used to track the plaintext record corresponding to each ciphertext record computed by the sender (see line 3.8:12). Before performing the AEAD decryption operation, the receiver consults this table to see if the ciphertext was previously computed by the sender (3.8:26). Because each nonce computed by the sender or receiver is unique, and assuming $AEAD$ is correct, this does not change the outcome of the experiment.

Experiment 0. Let $\mathbf{G}^0(D') = \mathbf{Real}^{\mathsf{win}}_W(D')$. One can easily verify that

$$\Pr\big[\,\mathbf{G}^0(D')\,\big] = \mathbf{Adv}^{\text{int-cs}}_A(D')\,. \tag{3.8}$$

Revision 0-1. Define procedure $\mathbf{G}^1$ from $\mathbf{G}^0$ by adding the code "$X \leftarrow \bot$" immediately after the flag bad gets set on line 3.8:29. We exhibit an INT-adversary D_0 such that

$$\Pr\big[\,\mathbf{G}^0(D')\,\big] \leq \Pr\big[\,\mathbf{G}^1(D')\,\big] + \mathbf{Adv}^{\text{int}}_{AEAD}(D_0)\,, \tag{3.9}$$

where D_0 is $O(t_A)$-time and makes q_s **Enc**-queries and q_r **Dec**-queries. Adversary D_0 simply runs $\mathbf{G}^0(D')$ with line 3.8:9 replaced with "$Y' \leftarrow \mathbf{Enc}(N, A, X)$" and line 3.8:28 replaced with "$X \leftarrow \mathbf{Dec}(N, A, Y')$". Flag bad gets set if a call of the form $\mathbf{Dec}(N, A, Y')$ returns a plaintext $X \neq \bot$. Because D_0 did not previously ask $\mathbf{Enc}(N, A, X)$, this implies that D_0 has won its game.

Revision 1-2. Now define procedure $\mathbf{G}^2$ from $\mathbf{G}^1$ by replacing the "$T[ct_0, A, Y'] \neq \bot$" condition on line 3.8:26 with "$Y \preceq s$". By way of showing that

$$\Pr\big[\,\mathbf{G}^1(D')\,\big] = \Pr\big[\,\mathbf{G}^2(D')\,\big]\,, \tag{3.10}$$

we argue that these conditions are equivalent. Let $\hat{s}$ denote the concatenation of channel fragments produced by the sender so far (see line 3.7:18) and let $\hat{t}$ denote the concatenation of channel fragments consumed by the receiver so far (3.7:23). Note that $s = \hat{s} \,\%\, \hat{t}$ whenever the channel is in-sync, i.e., when $sync = 1$. Fix $q \in [q_r]$ and run the experiment up to the point just before the q-th RECV-query returns. Suppose that $\widehat{sync} = 1$ and $Y \neq \varepsilon$ hold. At this point the receiver has consumed a sequence of non-empty channel fragments $Y_1, \ldots, Y_{ct_0-1}$ for which $\hat{t} = Y_1 \,\|\, \cdots \,\|\, Y_{ct_0-1}$. Because $\widehat{sync} = 1$ holds, for each $i \in [ct_0 - 1]$ there exist strings A_i, Y'_i such that $Y_i = A_i \,\|\, Y'_i$ and $T[i, A_i, Y'_i] \neq \bot$.

Suppose that $T[ct_0, A, Y'] \neq \bot$. Since $Y = A \,\|\, Y'$, it holds that $Y_1 \,\|\, \cdots \,\|\, Y_{ct_0-1} \,\|\, Y \preceq \hat{s}$. Thus, the channel is necessarily in-sync (i.e., $s = \hat{s} \,\%\, \hat{t}$), and so $Y \preceq s$. Now suppose that $Y \preceq s$ holds. This implies that the channel is in-sync (i.e., $s = \hat{s} \,\%\, \hat{t}$). In particular, there exist strings A, Y' such that $Y = A \,\|\, Y'$ and $T[ct_0, A, Y'] \neq \bot$. Moreover, by the construction of D' (see the proof of Lemma 3) it holds that $D'_2(\mathsf{parse}, Y) = (|Y|, |A|)$. Because the

```
procedure $\mathbf{Send}(msg, ctx)$:
1  var $N, A$ $\mathbf{str}$, $\gamma$ $\mathbf{any}$
2  $(X, \alpha) \leftarrow M_1(\textsf{MUX}, msg, ctx)$
3  if $X = \varepsilon$ then $N \leftarrow \varepsilon$
4  else
5      $N \leftarrow IV \oplus (0^{n-64} \,\|\, str_{64}(ct_1))$
6      $ct_1 \leftarrow ct_1 + 1$
7  $(A, \gamma) \leftarrow \mathcal{A}(\textsf{send ready}, |X|, \alpha)$
8  if $X = \varepsilon$ then ret $(X, \varepsilon, \gamma)$
9  $Y' \leftarrow AEAD_K(\textsf{ENC}, N, A, X)$
10 if $Y' = \bot \vee \mathcal{A}(\textsf{parse}, A \,\|\, Y') \neq (|A| + |Y'|, |A|)$ then
11     ret $(X, \varepsilon, \mathcal{A}(\textsf{invalid ptxt}))$
12 $T[ct_1, A, Y'] \leftarrow X$
13 ret $(X, A \,\|\, Y', \gamma)$
```

```
procedure $\mathbf{Recv}(c)$:
14 var $N, X$ $\mathbf{str}$, $u, v$ $\mathbf{int}$, $drop$ $\mathbf{bool}$, $\alpha, \gamma$ $\mathbf{any}$
15 $buf \leftarrow buf \,\|\, c$
16 $(u, v) \leftarrow \mathcal{A}(\textsf{parse}, buf)$; $(drop, \alpha) \leftarrow \mathcal{A}(\textsf{drop}, buf)$
17 $Y \leftarrow buf[:u]$; $buf \leftarrow buf \;\%\; Y$
18 if $Y = \varepsilon \vee drop = 1$ then $N \leftarrow \varepsilon$
19 else
20     $N \leftarrow IV \oplus (0^{n-64} \,\|\, str_{64}(ct_0))$
21     $ct_0 \leftarrow ct_0 + 1$
22 $X \leftarrow \varepsilon$; $\gamma \leftarrow \mathcal{A}(\textsf{recv ready}, |Y|, \alpha)$
23 if $(Y = \varepsilon \wedge \gamma \neq \bot) \vee \widehat{sync} = 0$ then ret $(Y, \bot, \bot, \gamma)$
24 else if $Y \neq \varepsilon$ then
25     $A \leftarrow Y[:v]$; $Y' \leftarrow Y \;\%\; A$
26     if $T[ct_0, A, Y'] \neq \bot$ then $X \leftarrow T[ct_0, A, Y']$
27     else
28         $X \leftarrow AEAD_K(\textsf{DEC}, N, A, Y')$
29         $bad \leftarrow 1$
30     if $X = \bot$ then
31         $\widehat{sync} \leftarrow 0$
32         ret $(Y, \bot, \bot, \mathcal{A}(\textsf{invalid ctxt}))$
33 $(msg, ctx, \gamma) \leftarrow M_0(\textsf{DEMUX}, X)$
34 ret $(Y, msg, ctx, \gamma)$
```

Figure 3.8: Procedures **Send** and **Recv** for the proofs of Claim 3 and Claim 4.

receiver computes the record boundaries just as the sender does, it must be that $T[ct_0, A, Y'] \neq \bot$ holds on line 3.8:26.

Experiment 2. Observe that procedure $\mathbf{G}^1$ sets $\widehat{sync} \leftarrow 0$ (line 3.8:31) if it processes a channel fragment Y for which $Y \not\preceq s$. Thus, any query that sets $sync \leftarrow 0$ also sets $\widehat{sync} \leftarrow 0$. It follows that

$$\Pr\big[\, \mathbf{G}^2(D') \,\big] = 0\,. \tag{3.11}$$

This completes the proof of Claim 3. □

A similar argument allows us to relate the SIM-STAT security of Λ to the SIM-mSTAT security of Mux.

Claim 4. For every t_S-time simulator S there exist $O(t_A)$-time adversaries C and D_1 and $O(t_S + t_A)$-time simulator S' for which

$$\mathbf{Adv}^{\text{sim-stat}}_{\Lambda}(C', S') \leq \mathbf{Adv}^{\text{sim-mstat}}_{Mux}(C, S) + \mathbf{Adv}^{\text{int}}_{AEAD}(D_1)\,,$$

where C makes at most q_r **Demux**-queries and D_1 at most q_s **Enc**-queries and at most q_r **Dec**-queries.

Proof. Let $S' = \mathbf{Sim}'(S, AEAD)$ as specified in Figure 3.10. Let world $W = \mathbf{World}(S, AEAD, Mux)$ and world $W' = \mathbf{World}'(S', S, AEAD, Mux)$ be as specified in Figure 3.9. Worlds W and W' were obtained from the SIM-STAT experiment for Λ by conditioning on the outcome of the challenge bit b (cf. Figure 3.2). World W corresponds to the case where RECV-queries are answered by the receiver (i.e., $b = 1$), and W' corresponds to the case where RECV-queries are answered by simulator S' (i.e, $b = 0$). There exists a $O(t_A)$-time adversary C'' for which

$$\mathbf{Adv}^{\text{sim-stat}}_{\Lambda}(C', S') = \Pr\big[\, \mathbf{Real}^{\mathsf{win}}_{W}(C'') \,\big] - \Pr\big[\, \mathbf{Real}^{\mathsf{win}}_{W'}(C'') \,\big] \tag{3.12}$$

and C'' has the same query complexity as C'. Adversary C'' simply runs C', answering its oracle queries using its own oracles in the natural way. When C'' asks $(\textsf{GUESS}, d\ \mathbf{bool})$ of its main interface, adversary C' halts and outputs d. In doing so, adversary C'' makes as many SEND- and RECV-queries as C'. Moreover, its runtime is $O(t_A)$ because C' is $O(t_A)$-time.

Experiments 0 and 3. Define $\mathbf{G}^0(C'') = \mathbf{Real}^{\mathsf{out}}_{W}(C'')$ and $\mathbf{G}^3(C'', S') = \mathbf{Real}^{\mathsf{out}}_{W'}(C'')$ so that

$$\mathbf{Adv}^{\text{sim-stat}}_{\Lambda}(C', S') = \Pr\big[\, \mathbf{G}^0(C'') \,\big] - \Pr\big[\, \mathbf{G}^3(C'', S') \,\big]\,. \tag{3.13}$$

```
spec World $\boxed{\textbf{World}'}$:
1   var $\boxed{S',}$ $S, AEAD, Mux, M_1, M_0$ object
2   var $bad, sync, \widehat{sync}$, bool, $K, IV, s, \hat{s}, \hat{t}$ str
3   var $T$ table, $buf$ str, $ct_1, ct_0$ int
4
5   op (SETUP):
6       $S$(SETUP); $Mux$(SETUP)
7       $M_1 \leftarrow M_0 \leftarrow Mux$
8       $init, bad \leftarrow 0$; $sync, \widehat{sync} \leftarrow 1$; $T \leftarrow [\,]$
9       $s, \hat{s}, \hat{t}, buf \leftarrow \varepsilon$; $ct_1, ct_0 \leftarrow 0$
10
11  op (1, INIT):
12      if $init \neq 1$ then $init \leftarrow 1$
13      $(K, IV) \twoheadleftarrow \mathcal{K} \times \{0,1\}^n$
14  op$^{-,\mathcal{A}}$ (1, SEND, $msg, ctx$ str):
15      if $init \neq 1$ then ret $(\bot, \bot)$
16      $(\_, c, \gamma) \leftarrow$ Send$(msg, ctx)$
17      $s \leftarrow s \,\|\, c$; $\hat{s} \leftarrow \hat{s} \,\|\, c$
18      ret $(c, \gamma)$
19
20  op$^{-,\mathcal{A}}$ (1, RECV, $c$ str):
21      var $\gamma$ any
22      if $init \neq 1$ then ret $(\bot, \bot)$
23      $(Y, \_, \_, \gamma) \leftarrow$ Recv$(c)$; $\hat{t} \leftarrow \hat{t} \,\|\, Y$
24      if $sync = 1 \wedge Y \preceq s$ then $s \leftarrow s \,\%\, Y$
25      else $sync \leftarrow 0$
26      $\boxed{\gamma \leftarrow S'^{\mathcal{A}}(\hat{s}, c)}$
27      ret $\gamma$
```

Figure 3.9: Worlds $W = \textbf{World}(S, AEAD, Mux)$ and $W' = \textbf{World}'(S', S, AEAD, Mux)$ for the proof of Claim 4. Procedures **Send** and **Recv** are defined in Figure 3.8.

In other words, procedure $\mathbf{G}^0$ (resp. $\mathbf{G}^3$) runs adversary C'' in world W (resp. W') and returns its output. We proceed by rewriting procedure $\mathbf{G}^0$ so that it is functionally equivalent to $\mathbf{G}^3$.

Revision 0-1. First, define $\mathbf{G}^1$ from $\mathbf{G}^0$ by modifying the definition of procedure **Recv** as shown in the left panel of Figure 3.10. This code was obtained by modifying procedure **Recv** as described in revisions 0-1 and 1-2 in the proof of Claim 3. Namely, referring to Figure 3.8: the statement "$X \leftarrow \bot$" was added immediately after *bad* is set on line 3.8:29; and the expression "$T[ct_0, A, Y'] \neq \bot$" on line 3.8:26 was replaced with "$Y \preceq s$". By the same argument that yielded Eq. (3.9) and Eq. (3.10), there exists an INT-adversary D_1 for which

$$\Pr\left[\,\mathbf{G}^0(C'')\,\right] \le \Pr\left[\,\mathbf{G}^1(C'')\,\right] + \mathbf{Adv}^{\text{int}}_{AEAD}(D_1)\,, \tag{3.14}$$

where D_1 is $O(t_A)$-time and makes q_s **Enc**-queries and q_r **Dec**-queries.

Revision 1-2. Define $\mathbf{G}^2$ from $\mathbf{G}^1$ by modifying **Recv** as shown in Figure 3.10. The substantive change is to replace execution of the stream multiplexer (3.10:21-22) with execution of the SIM-mSTAT-simulator S (3.10:23). Because S operates on the length of the pre-channel fragment X rather than X itself, the procedure has been modified so that X is not explicitly computed. We exhibit a SIM-mSTAT-adversary C for which

$$\mathbf{Adv}^{\text{sim-mstat}}_{Mux}(C, S) = \Pr\left[\,\mathbf{G}^1(C'')\,\right] - \Pr\left[\,\mathbf{G}^2(C'')\,\right]. \tag{3.15}$$

Adversary C runs $\mathbf{G}^1(C'')$, replacing "$(msg, ctx, \gamma) \leftarrow Mux(\text{DEMUX}, X)$; ret (Y, msg, ctx, γ)" with "ret $(Y, \bot, \bot, \mathbf{Demux}(X))$". When C'' halts, adversary C halts and outputs whatever C'' output. Note that C has the same runtime as C'' and makes at most q_r **Demux**-queries.

Experiment 2. Observe that the computation of procedure **Recv** in experiment $\mathbf{G}^2$ depends only on the channel fragment c and the string s that records the ciphertext stream output by the sender that has not yet been consumed by the receiver (cf. lines 3.9:24-25). The latter can be computed from the sender-produced stream $\hat{s}$ and the receiver-consumed $\hat{t}$, both of which are available to the simulator S' in experiment $\mathbf{G}^3$. Indeed, the specification of S' (right panel of Figure 3.10) was obtained by simplifying the code for **Recv** in $\mathbf{G}^2$. It is not difficult to verify that the behavior of the RECV-operator in $\mathbf{G}^2$ and $\mathbf{G}^3$ are equivalent. We conclude that

$$\Pr\left[\,\mathbf{G}^2(C'')\,\right] = \Pr\left[\,\mathbf{G}^3(C'', S')\,\right]. \tag{3.16}$$

Finally, we comment on the runtime of simulator S'. Each of its operators runs S at most once; otherwise, its computations are linear in the length of its inputs. By construction (see Lemma 3), these are bounded by the runtime of A because its runtime includes the time needed to evaluate its oracle queries. It follows that the runtime of simulator S' is $O(t_S + t_A)$. This completes the proof of Claim 4. □

```
procedure $\mathbf{Recv}(c)$:                                    $\mathbf{G}^1$ $\boxed{\mathbf{G}^2}$
var $N$, $X$ str, $u, v$, $\boxed{x}$ int, $drop$ bool, $\alpha, \gamma$ any
$buf \leftarrow buf \,\|\, c$
$(u, v) \leftarrow \mathcal{A}(\text{parse}, buf)$; $(drop, \alpha) \leftarrow \mathcal{A}(\text{drop}, buf)$
$Y \leftarrow buf[{:}u]$; $buf \leftarrow buf \,\%\, Y$
if $Y = \varepsilon \vee drop = 1$ then $N \leftarrow \varepsilon$
else
    $N \leftarrow IV \oplus (0^{n-64} \,\|\, str_{64}(ct_0))$
    $ct_0 \leftarrow ct_0 + 1$
$X \leftarrow \varepsilon$ $\boxed{x \leftarrow 0}$; $\gamma \leftarrow \mathcal{A}(\text{recv ready}, |Y|, \alpha)$
if $(Y = \varepsilon \wedge \gamma \neq \perp) \vee \widehat{sync} = 0$ then ret $(Y, \perp, \perp, \gamma)$
else if $Y \neq \varepsilon$ then
    $A \leftarrow Y[{:}v]$; $Y' \leftarrow Y \,\%\, A$
    if $Y \preceq s$ then
        $X \leftarrow T[ct_0, A, Y']$ $\boxed{x \leftarrow AEAD(\text{MLEN}, |Y'|)}$
    else
        $X \leftarrow AEAD_K(\text{DEC}, N, A, Y')$
        $bad \leftarrow 1$; $X \leftarrow \perp$
    if $X = \perp$ then
        $\widehat{sync} \leftarrow 0$
        ret $(Y, \perp, \perp, \mathcal{A}(\text{invalid ctxt}))$
$(msg, ctx, \gamma) \leftarrow M_0(\text{DEMUX}, X)$
ret $(Y, msg, ctx, \gamma)$
$\boxed{\text{ret } (Y, \perp, \perp, S(x))}$
```

```
spec $\mathbf{Sim}'$:
var $S, AEAD$ object, $buf, \hat{s}, \hat{t}$ str, $\widehat{sync}$ bool
op $(\text{SETUP})$:
    $S(\text{SETUP})$; $buf, \hat{s}, \hat{t} \leftarrow \varepsilon$; $\widehat{sync} \leftarrow 1$
op$^{\mathcal{A}}$ $(\hat{s}, c$ str$)$:
    $s \leftarrow \hat{s} \,\%\, \hat{t}$
    $(Y, \gamma) \leftarrow \mathbf{Recv}(s, c)$
    $\hat{t} \leftarrow \hat{t} \,\|\, Y$
    ret $\gamma$

procedure $\mathbf{Recv}(s, c)$:
var $u, v, x$ int, $\alpha, \gamma$ any
$buf \leftarrow buf \,\|\, c$
$(u, v) \leftarrow \mathcal{A}(\text{parse}, buf)$; $(\_, \alpha) \leftarrow \mathcal{A}(\text{drop}, buf)$
$Y \leftarrow buf[{:}u]$; $buf \leftarrow buf \,\%\, Y$
$x \leftarrow 0$; $\gamma \leftarrow \mathcal{A}(\text{recv ready}, |Y|, \alpha)$
if $(Y = \varepsilon \wedge \gamma \neq \perp) \vee \widehat{sync} = 0$ then ret $(Y, \gamma)$
else if $Y \neq \varepsilon$ then
    if $Y \preceq s$ then $x \leftarrow AEAD(\text{MLEN}, u - v)$
    else $\widehat{sync} \leftarrow 0$; ret $(Y, \mathcal{A}(\text{invalid ctxt}))$
ret $(Y, S(x))$
```

Figure 3.10: Definitions for the proof of Claim 4. Left: Definition of procedure **Recv** called by $\mathbf{G}^0$ and $\mathbf{G}^1$. Right: Specification of simulator $S' = \mathbf{Sim}'(S, AEAD)$.

To complete the proof, we consolidate the two INT-CS terms into one. Namely, define INT-adversary D by flipping a coin $b \twoheadleftarrow \{0,1\}$ and running D_b, where D_1 is given by Claim 3 and D_0 is given by Claim 4. Then for each $b \in \{0,1\}$ it holds that

$$\mathbf{Adv}^{\text{int}}_{AEAD}(D_b) \leq 2\mathbf{Adv}^{\text{int}}_{AEAD}(D)\,. \tag{3.17}$$

This completes the proof. ■

Remark 4. The restrictions on the adversary in Theorem 2 were not made in the original version of PS18. Some restrictions are necessary, however, as we are unable to prove SIM-STAT security of the channel for arbitrary adversaries. Indeed, this indicates that an error was made in the original proof. (The full version of the paper has been revised in order to correct the error; see [116, Theorem 4].) We remark that Rogaway and Stegers make a similar restriction in their proof of security of the NSL2 protocol (cf. [130, Theorem 1]). These restrictions are somewhat technical in nature, in the sense that they are necessary to prove security but do not impede a rigorous analysis of TLS 1.3. □

3.3 Discussion

The formal model presented in this chapter (§3.1) advances the theory of secure channels by considering the multiplexing of many data streams over a single channel, an essential feature of modern protocols like TLS. By viewing data as a stream, our syntax and execution environment for channels accurately reflects how their APIs process inputs. This leads to a strong attack model in which many of the details left unspecified by the TLS standard are crucial for security. We prove that the TLS 1.3 record layer meets our privacy goal, but with two caveats. First, whether the record layer hides the length, content, or type of input streams depends crucially on details left unspecified by the standard. Second, the optional behavior of the draft standard does not allow us to prove integrity of the ciphertext stream. However, we suggest a simple change to the standard so that it provably does. (This change was adopted in the final standard [123].)

Our formal specification (Figure 3.5) provides a degree of robustness to changes to the standard that might affect the record layer. For example, the record boundaries are an especially flexible aspect of the specification, since future TLS extensions might add new content types. Our treatment handles, quite seamlessly, the impact on record boundaries that such extensions would induce. Still, there are features of the standard that our analysis does not (directly) account for.

- Our formal treatment does not address negotiation of the AEAD scheme during the handshake protocol. Consequentially, any agility [5] issues that arise as a result are not accounted for. It should be noted that TLS 1.3 takes steps to ensure that keys used by different ciphersuites (i.e., AEAD schemes) are independent: in particular, key derivation is bound to the selected ciphersuite. (We provide more details on the key schedule in §4.2.) However, our specification of the record layer does not capture key derivation and instead assumes ideal distribution of the key (and IV) used to protect the channel. (See the GEN-operator on lines 3.5:5-6.)
- Nor does our specification account for key changes, which, as we mentioned before, occur throughout the normal course of the protocol. This means that the proved security properties only apply to a single "phase" of the protocol (i.e., early data, handshake data, or application data). This leaves open the question of whether the concatenation of streams sent across key changes are secure.
- Translating the specification of the record layer in order to account for these things would bring to the fore a variety of protocol-level side-effects that do not come up in the analysis of Theorem 2. For example, explicitly modeling key derivation would also require us to account for other uses of the initial key material (e.g., for exporting of pre-shared keys).

Accounting for these discrepancies would require revising the specification of the record layer (Figure 3.5), and so too the proof of Theorem 2. The translation framework of Chapter 2 affords a potential shortcut in which one extends Theorem 2 to the revised specification by reasoning directly about the translation itself. The remainder of this dissertation exhibits two applications of this approach: one to the translation of a scheme's specification (Chapter 4); and another to the translation of its execution environment (Chapter 5).

Chapter 4

Protocol Translation

The effort to standardize TLS 1.3 [123] was remarkable in that it leveraged provable security as part of the standardization process [113]. Perhaps the most influential of these works was Krawczyk and Wee's OPTLS protocol [93], which served as the basis for an early draft of the TLS 1.3 handshake. Core features of OPTLS are recognizable in the final standard, but TLS 1.3 is decidedly not OPTLS. As is typical of the standardization process, protocol details were modified in order to address deployment and operational desiderata (cf. [113, §4.1]). Naturally, this raises the question of what, if any, of the proven security that supported the original AKE protocol is inherited by the standard.

The objective of this chapter is to answer a general version of this question: given a reference protocol $\tilde{\mathit{\Pi}}$ (e.g., OPTLS), what is the cost, in terms of concrete security [21], of translating $\tilde{\mathit{\Pi}}$ into some real protocol $\mathit{\Pi}$ (e.g., TLS 1.3) with respect to the security notion(s) targeted by $\tilde{\mathit{\Pi}}$? This question is moot for OPTLS/TLS 1.3, of course, since the standard has undergone continual analysis in the time since Krawczyk and Wee's initial proposal. Still, cryptographic standards continue to exhibit behavior that deviates from the "cleanroom" analysis of the protocols from which they are derived.

A more recent standardization effort provides an illustrative case study. At the time of writing, the CFRG[1] was in the midst of selecting a portfolio of password-authenticated key-exchange (PAKE) protocols [27] to recommend to the IETF[2] for standardization. (The objective of a PAKE protocol is to securely establish a key between two parties who initially only share a low-entropy secret, such as a password. We discuss the intended security model for this class of protocols below.) Among the CFRG's selection criteria [139] is the suitability of the PAKE for integration into existing protocols. In the case of TLS, the goal would be to standardize an extension [123, §4.2] that specifies the usage of the PAKE in the TLS handshake. This would enable defense-in-depth for applications in which passwords are available for authentication, and sole reliance on the web PKI[3] for authentication is undesirable (or impossible). Thus, the extension ($\mathit{\Pi}$) should provide at least the same level of security as the PAKE itself ($\tilde{\mathit{\Pi}}$), with perhaps a modest loss in concrete security.

Indifferentiable Execution of eCK-Protocols. We define security for PAKE in the extended Canetti-Krawczyk (eCK) model of LaMacchia et al. [99], a simple, yet powerful model for the study of authenticated key exchange. The eCK model specifies both the execution environment of the protocol (i.e., how the adversary interacts with it) and its intended goal (i.e., key indistinguishability [30]) in a single security experiment. Our treatment breaks this abstraction boundary.

Recall from Chapter 2 that for any transcript predicate ψ, game G, and systems X and $\tilde{X}$, we can argue that X is G^{ψ}-secure (Def. 8) by proving that X is ψ-indifferentiable from $\tilde{X}$ (Def. 3) and assuming $\tilde{X}$ itself is G^{ψ}-secure. In this chapter, the system specifies the execution environment of a cryptographic protocol for which the game defines security. In §4.1 we specify a system $\mathbf{eCK}(\mathit{\Pi})$ that formalizes the execution of protocol $\mathit{\Pi}$ in the eCK model. Going up a level of abstraction, running an adversary A in world $W = \mathbf{Wo}(G, \mathbf{eCK}(\mathit{\Pi}))$ in the MAIN^{ψ} experiment (Def. 1) lets A execute $\mathit{\Pi}$ via W's auxiliary interface and "play" the game G via W's main interface. The environment **eCK**

[1] Cryptography Forum Research Group.

[2] Internet Engineering Task Force.

[3] Public-Key Infrastructure.

surfaces information about the state of the execution environment, which G uses to determine if A wins. Finally, transcript predicate ψ is used to determine if the attack is valid based on the sequence of W_1- and W_2-queries made by A.

We specify a game $G = \mathbf{Key\text{-}Ind}(f, k)$ that formalizes the security goal of the eCK model. This is the standard notion of key indistinguishability [30] for a particular notion of session "freshness", as formalized by the parameter f. In fact, a wide variety of key-indistinguishability notions in the literature can be captured this way [30, 27, 57, 71], as well as other security goals [74, 41, 3]. This is made possible by "splitting up" the execution environment and security goal, as we have done.

Case Study. In §4.2 we propose and prove secure a PAKE extension for TLS, which integrates the SPAKE2 protocol [4] into the handshake. Its design is based on existing Internet-Drafts [16, 98] and discussions on the CFRG mailing list [44, 144]. Our analysis unearths some interesting and subtle design issues. First, existing proposals effectively replace the DH key-exchange with execution of the PAKE, feeding the PAKE's output into the key schedule instead of the DH shared secret. As we will discuss, whether this usage is "safe" depends on the particular PAKE and its security properties. Second, our extension adopts a "fail closed" posture, meaning if negotiation of the PAKE fails, then the client and server tear down the session. Existing proposals allow them to "fail open" by falling back to standard certificate-only authentication. We do not see a way to account for this behavior in the proof of Theorem 3, at least not without relying on certificates. But this in itself is interesting, as it reflects the practical motivation of the extension: it makes little sense to fail open if one's goal is to reduce reliance on the web PKI.

Related work. The SPAKE2 protocol was first proposed and analyzed in 2005 by Abdalla and Pointcheval [4], who sought a simpler alternative to the seminal encrypted key-exchange (EKE) protocol of Bellovin and Merritt [35]. Given the CFRG's recent interest in SPAKE2 (and its relative SPAKE2+ [59]), there has been a respectable amount of recent security analysis. This includes concurrent works by Abdalla and Barbosa [1] and Becerra et al. [20] that consider the forward secrecy of (variants of) SPAKE2, a property that Abdalla and Pointcheval did not address. Victor Shoup [137] provides an analysis of a variant of SPAKE2 in the UC framework [53], which has emerged as the de facto setting for studying PAKE protocols (cf. OPAQUE [86] and (Au)CPace [81]). Shoup observes that the usual notion of UC-secure PAKE [56] cannot be proven for SPAKE2, since the protocol on its own does not provide key confirmation. Indeed, many variants of SPAKE2 that appear in the literature add key confirmation in order to prove it secure in a stronger adversarial model (cf. [20, §3]).

A recent work by Skrobot and Lancrenon [138] characterizes the general conditions under which it is secure to compose a PAKE protocol with an arbitrary symmetric key protocol (SKP). While their object of study is similar to ours—a PAKE extension for TLS might be viewed as a combination of a PAKE and the TLS record layer protocol—our security goals are different, since in their adversarial model the adversary's goal is to break the security of the SKP.

4.1 eCK-Protocols

We begin this section by formalizing the syntax and execution environment for eCK-protocols (§4.1.1). We then recall the standard notion of key indistinguishability and formalize it in our framework (§4.1.2). Finally, we end this section by discussing the relationship between our framework and the formal treatment of downgrade resilience of Bhargavan et al. [41] (§4.1.3).

4.1.1 Syntax And Execution Environment

The eCK model was introduced by LaMacchia et al. [99] in order to broaden the corruptive powers of the adversary in the Canetti-Krawczyk setting [57]. The pertinent change is to restrict the class of protocols to those whose state is deterministically computed from the player's static key (i.e., its long-term secret),[4] ephemeral key (i.e., the per-session randomness),[5] and the sequence of messages received so far. This results in a far simpler formulation of

[4] LaMacchia et al. use the term "long-term" key; Bellare et al. use the term "long-lived key" [30, 27]. We use the term "static key", following the Noise protocol framework [120], so that the "s" in "sk" may stand for "static".

[5] This is similar to the classical Bellare-Rogaway (BR) setting [30]. However, in BR the randomness consumed by a session may be unbounded, whereas in eCK, the randomness is sampled from a finite set.

```
spec eCK: // $\mathcal{A}$ points to $A_2$; $\mathcal{R}$ to resources
 1  var $\mathit{\Pi}$ object, $r$ int
 2  var $pk, sk, ek, \alpha, \pi$ table; $atk$ any
 3  op (SETUP):
 4      $\mathit{\Pi}$(SETUP); $r \leftarrow \mathit{\Pi}$(MOVES)
 5      $pk, sk, ek, \alpha, \pi \leftarrow []$; $atk \leftarrow ()$
 6
 7  // Main interface
 8  op$^{\mathcal{A},\mathcal{R}}$ (1, INIT):
 9      $(pk, sk) \leftarrow \mathit{\Pi}^{\mathcal{A},\mathcal{R}}$(SGEN); ret $pk$
10  op$^{\mathcal{A},\mathcal{R}}$ (1, GAME ST, $x, s, i$ str):
11      ret $\mathit{\Pi}^{\mathcal{A},\mathcal{R}}$(GAME ST, $x, i, \pi^i_s$)
12  op (1, ATTACK ST): ret $atk$
13
14  // Auxiliary interface
15  op (2, PK, $i$ str): ret $pk_i$
16  op (2, SK, $i$ str):
17      $atk \leftarrow atk$ . (SK, $i$); ret $sk_i$
18  op (2, EK, $s, i$ str):
19      $atk \leftarrow atk$ . (EK, $s, i$); ret $ek^i_s$
20  op$^{\mathcal{A},\mathcal{R}}$ (2, INIT, $s, i$ str, $a$ any):
21      Init(active, $s, i, a$)
22  op$^{\mathcal{A},\mathcal{R}}$ (2, SEND, $s, i$ str, $in$ any):
23      ret Send($s, i, in$)
24  op$^{\mathcal{A},\mathcal{R}}$ (2, EXEC, $s_1, i_1, s_0, i_0$ str, $a_1, a_0$ any):
25      Init(passive, $s_1, i_1, a_1$)
26      Init(passive, $s_0, i_0, a_0$)
27      $out \leftarrow \perp$; $tr \leftarrow ()$
28      for $j \leftarrow 1$ to $r + 1$ do $\gamma \leftarrow j \pmod 2$
29          $out \leftarrow$ Send($s_\gamma, i_\gamma, out$)
30          $tr \leftarrow tr$ . $out$
31      ret $tr$

procedure Init($t, s, i, a$): // $t \in$ {active, passive}
32  $ek^i_s \leftarrow \mathit{\Pi}^{\mathcal{A},\mathcal{R}}$(EGEN, $i, a$)
33  $\alpha^i_s \leftarrow a$; $\pi^i_s \leftarrow \perp$
34  $atk \leftarrow atk$ . $(t, s, i)$

procedure Send($s, i, in$):
35  $(\pi^i_s, out) \leftarrow \mathit{\Pi}^{\mathcal{A},\mathcal{R}}$(SEND, $i, sk_i, ek^i_s, \alpha^i_s, \pi^i_s, m$)
36  ret $out$
```

Figure 4.1: Execution environment for two-party eCK-protocols.

session-state compromise. We embellish the syntax by providing the party with an *initial input* at the start of each session, allowing us to capture features like per-session configuration [41].

Definition 15 (Protocols)**.** An *(eCK-)protocol* is a halting, stateless object $\mathit{\Pi}$, with an associated finite set of *identities* $\mathcal{I} \subseteq \{0,1\}^*$, that exports the following operators:

- (SGEN)-(pk, sk **table**): generates the static key and corresponding public key of each party so that (pk_i, sk_i) is the public/static key pair of party $i \in \mathcal{I}$.
- (EGEN, i **str**, α **any**)-(ek **any**): generates an ephemeral key ek for party i with input α. The ephemeral key constitutes the randomness used by the party in a given session.
- (SEND, i **str**, sk, ek, α, π, in **any**)-(π', out **any**): computes the outbound message out and updated state π' of party i with static key sk, ephemeral key ek, input α, session state π, and inbound message in. This operator is deterministic.
- (MOVES)-(r **int**): indicates the maximum number of moves (i.e., messages sent) in an honest run of the protocol. This operator is deterministic. □

The execution environment for eCK-protocols is specified by **eCK** in Figure 4.1. The environment stores the public/static keys of each party (tables pk and sk) and the ephemeral key (ek), input (α), and current state (π) of each session. As usual, the adversary is responsible for initializing and sending messages to sessions, which it does by making queries to the auxiliary interface (4.1:14-31). Each session is identified by a pair of strings (s, i), where s is the *session index* and i is the identity of the party incident to the session. The auxiliary interface exports the following operators:

- (INIT, s, i **str**, a **any**): initializes session (s, i) on input a by setting $\alpha^i_s \leftarrow a$ and $\pi^i_s \leftarrow \perp$. A session initialized in this way is said to be under *active attack* because the adversary controls its execution.
- (SEND, s, i **str**, in **any**)-(out **any**): sends message in to a session (s, i) under active attack. Updates the session state π^i_s and returns the outbound message out.
- (EXEC, s_1, i_1, s_0, i_0 **str**, a_1, a_0 **any**)-(tr **any**): executes an honest run of the protocol for initiator session (s_1, i_1) on input a_1 and responder session (s_0, i_0) on input a_0 and returns the sequence of exchanged messages tr. A session initialized this way is said to be under *passive attack* because the adversary does not control the protocol's execution.

- $(\text{PK}, i\ \mathbf{str})$-$(pk\ \mathbf{any})$, $(\text{SK}, i\ \mathbf{str})$-$(sk\ \mathbf{any})$, and $(\text{EK}, s, i\ \mathbf{str})$-$(ek\ \mathbf{any})$: returns, respectively, the public key of party i, the static key of party i, and the ephemeral key of session (s, i).

Whenever the protocol is executed, it is given access to the adversary's auxiliary interface (see interface oracle $\mathcal{A}$ on lines 4.1:9, 11, 32, and 35). This allows us to formalize security goals for protocols that are only partially specified [130]. In world $\mathbf{Wo}(G, X)$, system $X = \mathbf{eCK}(\mathit{\Pi})$ relays $\mathit{\Pi}$'s $\mathcal{A}$-queries to G: usually game G will simply forward these queries to the adversary, but the game must explicitly define this. (See §4.1.2 for an example.)

The attack state (atk) records the sequence of actions carried out by the adversary. Specifically, it records whether each session is under active or passive attack (4.1:34), whether the adversary knows the ephemeral key of a given session (4.1:19), and which static keys are known to the adversary (4.1:17). These are used by the game to decide if the adversary's attack was successful. In addition, the game is given access to the *game state*, which surfaces any artifacts computed by a session that are specific to the intended security goal: examples include the session key in a key-exchange protocol, the session identifier (SID) or partner identifier (PID) [27], or the negotiated mode [41]. The game state is exposed by the protocol's GAME␣ST-interface (e.g., lines 4.3:8-13). All told, the main interface (4.1:7-12) exports the following operators:

- (INIT)-$(pk\ \mathbf{any})$: initializes each party by running the static key generator and returns the table of public keys pk.
- (ATTACK ST)-$(atk\ \mathbf{any})$: returns the attack state atk to the caller.
- $(\text{GAME ST}, x, s, i\ \mathbf{str})$-$(val\ \mathbf{any})$: provides access to the game state.

Attack Validity. For simplicity, our execution environment allows some behaviors that are normally excluded in security definitions. Namely, (1) the adversary might initialize a session before the static keys have been generated, or try to generate the static keys more than once; or (2) the adversary might attempt to re-initialize a session already in progress. The first of these is excluded by transcript predicate ϕ_{init} and the second by ϕ_{sess}, both defined below.

Definition 16 (Predicates ϕ_{init} and ϕ_{sess})**.** Let $\phi_{\text{init}}(tx) = 1$ if $|tx| \geq 1$, $tx_1 = (1, \text{INIT})$, and for all $1 < \alpha \leq |tx|$ it holds that $tx_\alpha \neq (1, \text{INIT})$. Let $\phi_{\text{sess}}(tx) = 0$ iff there exist $1 \leq \alpha < \beta \leq |atk|$ such that $atk_\alpha = (t_\alpha, s_\alpha, i_\alpha)$, $atk_\beta = (t_\beta, s_\beta, i_\beta)$, $(s_\alpha, i_\alpha) = (s_\beta, i_\beta)$, and $t_\alpha, t_\beta \in \{\text{passive}, \text{active}\}$, where atk is the attack state corresponding to transcript tx. □

Comparison to LaMacchia et al. Apart from the possibility of the protocol being only partially specified, there are a few minor differences between the execution environment described above and that of LaMacchia et al. These are inherited from recent iterations of the eCK model, in particular eCK^w/FPS of Cremers-Feltz [62] and FPS-PSK of Dowling-Paterson [71]. First, our experiment involves an explicit session index s used to distinguish between sessions (s, i) pertaining to the same party i. Our protocols do not operate on s, but solely on the conversation associated with the session (i.e., the session's state and the inbound message). Second, we do not allow the adversary to specify the parties' identities (that is, the set $\mathcal{I}$) or the public key of corrupted parties.[6] Third, our environment includes an explicit session-initialization operation, allowing us to model per-session configuration [41].

4.1.2 KEY-IND Security

In order to demonstrate how security properties for eCK-protocols are defined in our framework, in this section we specify a game that captures the standard notion of key indistinguishability. The game is defined so that freshness of the test session is a parameter, allowing us to capture in one experiment a large variety of corruption models and partnering notions [30, 27, 57], including the notion of LaMacchia et al.

A *freshness predicate* is a halting, functional object that exports a $(\_\ \mathbf{set}, (\_, \_\ \mathbf{str}))$-$\mathbf{bool}$-operator. Let f be a freshness predicate and let $k \geq 0$ be an integer. Let $\mathit{\Pi}$ be a protocol (Def. 15) and let $X = \mathbf{eCK}(\mathit{\Pi})$ be as specified in Figure 4.1. Let $G = \mathbf{Key\text{-}Ind}(f, k)$ be as specified in Figure 4.2. The objective of the adversary in world $\mathbf{Wo}(G, X)$ (illustrated in Figure 4.2) is to distinguish some session key computed by system X from a random, k-bit string.

[6] In theory, allowing the adversary to specify these things results in a stronger attack model, but we are unaware of an instance of this capability leading to an attack in practice.

```
spec Key-Ind:  // $\mathcal{X}$ points to $X_1$; $\mathcal{A}$ points to $A_2$.
1  var $f$ object, $k$ int
2  var $b, w, init, guess, test$ bool, $s^*, i^*$ any, $\mathcal{S}$ set
3  op (SETUP): $b \twoheadleftarrow \{0,1\}$; $w, init, guess, test \leftarrow 0$; $\mathcal{S} \leftarrow ()$
4  op (1, WIN): ret $w$
5  op$^{\mathcal{X},-}$ (1, INIT): if $init \neq 1$ then $init \leftarrow 1$; ret $\mathcal{X}$(INIT)
6  op$^{\mathcal{X},-}$ (1, REVEAL, $s, i$ str): $\mathcal{S} \leftarrow \mathcal{S} \cup \{(s,i)\}$; ret $\mathcal{X}$(GAME ST, key, $s, i$)
7  op$^{\mathcal{X},-}$ (1, TEST, $s, i$ str):
8      $K_1 \leftarrow \mathcal{X}$(GAME ST, key, $s, i$); $K_0 \twoheadleftarrow \{0,1\}^k$
9      if $test \neq 1 \wedge K_1 \neq \bot$ then $test \leftarrow 1$; $(s^*, i^*) \leftarrow (s,i)$; ret $K_b$
10 op$^{\mathcal{X},-}$ (1, GUESS, $d$ bool):
11     if $guess \neq 1$ then $guess \leftarrow 1$
12         $w \leftarrow (b = d) \wedge f^{\mathcal{X}}(\mathcal{S}, (s^*, i^*)) = 1$
13 op$^{-,\mathcal{A}}$ (2, $x$ any): ret $\mathcal{A}(x)$
```

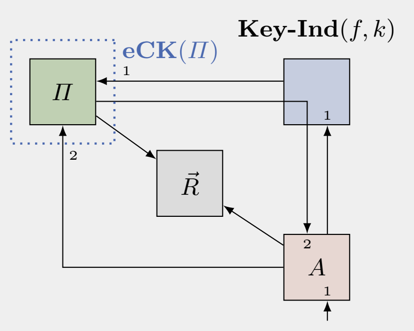

Figure 4.2: Left: Specification of the key-indistinguishability game **Key-Ind**(f,k). Right: Illustration of the real experiment for adversary A in world **Wo**(**Key-Ind**(f,k), **eCK**$(\mathit{\Pi})$) with resources $\vec{R}$.

During setup, the game chooses a uniform random bit b. After initializing X (4.2:5), the adversary concurrently executes a number of sessions of $\mathit{\Pi}$ by interacting with X's auxiliary interface. During its attack, the adversary may learn session keys by making REVEAL-queries to G (4.2:6), in addition to corrupting static and ephemeral keys via direct access to X. Eventually the adversary makes a (TEST, s, i **str**)-query to G (4.2:7-9): if the session key for session (s,i) is available, then it is set to K_1; and K_0 is set to random k-bit string. Finally, the query returns K_b. Some time after its TEST-query, the adversary makes a (GUESS, d **str**)-query to G, which sets the outcome of the game to be

$$w = (b = d) \wedge f^{\mathcal{X}}(\mathcal{S}, (s^*, i^*)) = 1\,, \tag{4.1}$$

where $\mathcal{X}$ denotes G's oracle for X_1, session (s^*, i^*) is the TEST-session, and $\mathcal{S}$ is the set of sessions whose session keys have been revealed. In English, the game G deems the adversary to have won if (1) it correctly guesses the challenge bit b and (2) the TEST-session is fresh according to f. The freshness predicate is given the set $\mathcal{S}$ and oracle access to $\mathcal{X}$ so that it may inspect the game and attack state to determine if the adversary is able to trivially compute the TEST-session key. The adversary makes at most one TEST-query and at most one GUESS-query. The outcome of the game is the value of w when the adversary halts (4.2:4).

Definition 17 (KEY-IND ^{f,k} security)**.** Let $\vec{R}$ be resources, A be an adversary, and $\mathit{\Pi}$, G, X, f, and k be as defined above. Define the KEY-IND ^{f,k} advantage of A in attacking $\mathit{\Pi}/\vec{R}$ as

$$\mathbf{Adv}^{\text{key-ind}^{f,k}}_{\mathit{\Pi}/\vec{R}}(A) := 2\mathbf{Adv}^{G^{\psi}}_{X/\vec{R}}(A) - 1\,,$$

where $\psi = \phi_{\text{init}} \wedge \phi_{\text{sess}}$ and ϕ_{init} and ϕ_{sess} are defined in Def. 16. □

The Role of the Freshness Predicate. The precise meaning of the term "freshness" varies in the literature, but is usually determined by two factors. The first is the corruption model dictating which values can be revealed to the adversary and when. This determines which security properties can be formalized for the protocol, e.g. (weak) perfect forward secrecy (PFS), resistance to key-compromise impersonation (KCI) attacks, and so on. (See Krawczyk [95] for discussion of these and other properties.) The second is the notion of partnered sessions. Partnering takes many forms in the literature, from matching conversations [30, 99, 62], to protocol-dependent "partnering functions" [31], to partnering by SIDs computed from the conversation [27, 41]. The most suitable notion of partnering depends on the number of parties in the protocol, who communicates with whom, and what are the protocol's goals.

4.1.3 Exercise: Downgrade-Secure Sub-Protocols

For a given game G and transcript predicate ψ, the G^{ψ}-security of an eCK-protocol $\mathit{\Pi}$ is captured by running an adversary in world **Wo**(G, **eCK**$(\mathit{\Pi})$) in the MAIN$^{\psi}$ experiment. Our syntax and execution environment are general enough to capture a variety of security properties for two-party protocols.[7] For any game G, we are able to make precise statements of the following form:

> Given that eCK-protocol $\tilde{\Pi}$ is G^{ψ}-secure, if the execution of eCK-protocol Π is ψ-indifferentiable from the execution of eCK-protocol $\tilde{\Pi}$, then Π is also G^{ψ}-secure.

This provides a way to argue that a protocol Π is secure by "lifting" the existing analysis of $\tilde{\Pi}$, rather than proving G^{ψ}-security of Π from scratch. This approach is not always available, however, since Π might be so different from $\tilde{\Pi}$ that indifferentiability does not hold. It is most useful when Π is related to $\tilde{\Pi}$ in some manner: perhaps Π was derived from $\tilde{\Pi}$ by modifying its specification, say, by adding or deleting some feature; in the next section, we consider a case where Π specifies the execution of $\tilde{\Pi}$ in a higher-level protocol. In general we will think of Π as being some "real" protocol whose security we hope follows from the "reference" protocol $\tilde{\Pi}$.

Another interpretation is that $\tilde{\Pi}$ is not a concrete protocol, but rather a "boiled down" version of the full protocol Π. In their formal treatment of downgrade-resilient AKE, Bhargavan et al. [41] adopt this view in their approach to taming the complexity of real-world standards. The first step is to extract from a protocol's specification (i.e., Π) a "sub-protocol" (i.e., $\tilde{\Pi}$) that captures the features that are essential to the security property being considered. Downgrade resilience is then proven for the sub-protocol and lifted to the full one by applying their "downgrade security lifting" theorem [41, Theorem 2], which transforms an attack against the full protocol into an attack against the sub-protocol. Intuitively, this theorem defines a set of full protocols whose downgrade security follows from the downgrade security of the sub-protocol; part of the job of the analyst is to ensure if the real protocol is in this set.

Loosely speaking, Bhargavan et al. define a protocol $\tilde{\Pi}$ to be a "sub-protocol" of Π if there exists an efficient simulator S such that Π and the "composition of S with $\tilde{\Pi}$" (cf. [41, Def. 11]) are information-theoretically indistinguishable from one another when run in the downgrade resilience experiment. They argue that if $\tilde{\Pi}$ is a sub-protocol for Π, then for every adversary A attacking Π there exists an adversary B attacking $\tilde{\Pi}$ that gets at least as much advantage.

Intuitively, the statement "the execution of Π is indifferentiable from the execution of $\tilde{\Pi}$" amounts to a computational analogue of "$\tilde{\Pi}$ is a sub-protocol of Π". In turn, our Lemma 1 and Lemma 2 imply a computational analogue of [41, Theorem 2] for eCK-protocols. Moreover, we can show that lifting applies to a wide variety of security goals, and not just downgrade resilience.

Proposition 3 (Generalization of [41, Theorem 2])**.** *Let ψ be a transcript predicate and let Π and $\tilde{\Pi}$ be eCK-protocols. Let $X = \mathbf{eCK}(\Pi)$ and $\tilde{X} = \mathbf{eCK}(\tilde{\Pi})$. For every game G, t_A-time, n.d. adversary A, and t_S-time simulator S there exist n.d. adversaries D and B for which*

$$\mathbf{Adv}_{X}^{G^{\psi}}(A) \leq \mathbf{Adv}_{\tilde{X}}^{G^{\psi}}(B) + \mathbf{Adv}_{X,\tilde{X}}^{\mathrm{indiff}^{\psi}}(D,S)\,,$$

where D is $O(t_A)$-time, and B is $O(t_A t_S)$-time.

Proof. The claim follows from a straight-forward application of Lemma 1 and Lemma 2. ■

4.2 Case Study: PAKE Extension for TLS 1.3

Existing proposals for PAKE extensions [143, 16] allow passwords to be used either in lieu of certificates or alongside them in order to "hedge" against failures of the web PKI. Barnes and Friel [16] propose a simple, generic extension for TLS 1.3 [123] (draft-barnes-tls-pake) that replaces the standard DH key-exchange with a 2-move PAKE. This straight-forward approach is, arguably, the best option in terms of computational overhead, modularity, and ease-of-implementation. Thus, our goal will be to instantiate draft-barnes-tls-pake with SPAKE2. We begin with an overview of the extension and the pertinent details of TLS. We then describe the SPAKE2 protocol and specify its usage in TLS. We end with our security analysis.

[7] One exception is ACCE-style games [85], as these require an operator that challenges the adversary to distinguish which of two messages is being encrypted. One could define an execution environment suitable for these, but we will not do so here.

4.2.1 Usage of PAKE with TLS 1.3 (draft-barnes-tls-pake)

The TLS handshake begins when the client sends its “ClientHello” message to the server. The server responds with its “ServerHello” followed by its parameters “EncryptedExtensions” and “CertificateRequest” and authentication messages “Certificate”, “CertificateVerify”, and “Finished”. The client replies with its own authentication messages “Certificate”, “CertificateVerify”, and “Finished”. The Hellos carry ephemeral DH key shares signed by the parties' Certificates, and the signatures are carried by the CertificateVerify messages. Each party provides key confirmation by computing a MAC over the handshake transcript; the MACs are carried by the Finished messages.

The DH shared secret is fed into the “key schedule” [123, §7.1] that defines the derivation of all symmetric keys used in the protocol. Key derivation uses the *HKDF* function [92], which takes as input a “salt” string, the “initial key material (IKM)” (i.e., the DH shared secret), and an “information” string used to bind derived keys to the context in which they are used in the protocol. The output is used as a salt for subsequent calls to *HKDF*.[8] The first call is $\mathit{salt} \leftarrow \mathit{HKDF}(0^k, \mathit{psk}, \textsf{derived})$, where $k \geq 0$ is a parameter of TLS called the hash length[9] and *psk* is the pre-shared key. (If available, otherwise $\mathit{psk} = 0^k$.) Next, the parties derive the client handshake-traffic key[10] $K_1 \leftarrow \mathit{HKDF}(\mathit{salt}, \mathit{dhe}, \mathit{info}_1)$, the server handshake-traffic key $K_0 \leftarrow \mathit{HKDF}(\mathit{salt}, \mathit{dhe}, \mathit{info}_0)$, and the session key $K \leftarrow \mathit{HKDF}(\mathit{salt}, \mathit{dhe}, \textsf{derived})$. Variable *dhe* denotes the shared secret. Each information string encodes both Hellos and a string that identifies the role of the key: $\textsf{c hs traffic}$ for the client and $\textsf{s hs traffic}$ for the server. The traffic keys are used for encrypting the parameter and authentication messages and computing the Finished MACs, and the session key is used for encrypting application data and computing future pre-shared keys.

Extensions. Protocol extensions are typically comprised of two messages carried by the handshake: the *request*, carried by the ClientHello; and the *response*, carried by the ServerHello or by one of the server's parameter or authentication messages. Usually the request indicates support for a specific feature and the response indicates whether the feature will be used in the handshake. In draft-barnes-tls-pake, the client sends the first PAKE message in an extension request carried by its ClientHello; if the server chooses to negotiate usage of the PAKE, then it sends the second PAKE message as an extension response carried by its ServerHello. When the extension is used, the PAKE specifies the values of *psk* and *dhe* in the key schedule.

At first brush, it may seem “obvious” that the security of the extension follows immediately from the security of the PAKE, since the PAKE is run without modification. There are two important points to note here. The first is that the extension is underspecified: the output of a PAKE is generally a single session key, so it is up to the implementer to decide how the session key is mapped to the inputs of the key schedule (i.e., *psk* and *dhe*). The second point is that the PAKE is not only used to derive the session key (used to protect application data), but also to encrypt handshake messages and compute MACs. As a result, whether this usage is secure or not depends on the concrete protocol and how it is implemented in the extension.

4.2.2 The SPAKE2 Protocol

Designed by Abdalla and Pointcheval [4], SPAKE2 (pronounced “S-PAKE-TWO”) is simpler than the other PAKE candidates [86, 81], which makes it a good target for use in a higher-level protocol [144]. It uses just two primitives: a prime-order, cyclic group $\mathbb{G} = (\mathcal{G}, \cdot)$ and a hash function $H : \mathcal{T} \to \{0,1\}^k$, where $\mathcal{T} = \{0,1\}^* \times \{0,1\}^* \times \mathcal{G} \times \mathcal{G} \times \mathbb{Z}_{|\mathcal{G}|} \times \mathcal{G}$.The client and server share a password $sk \in \mathbb{Z}_{|\mathcal{G}|}$, which serves as the static key; and each of the parties has an ephemeral key drawn uniformly from $\mathbb{Z}_{|\mathcal{G}|}$. The protocol has public parameters $N_1, N_0 \in \mathcal{G}$ chosen during a trusted setup phase that precedes the protocol's execution.[11]

The 2-move protocol is based on EKE [35]. The first message is sent by the client and consists of the client's identity c and password-masked key-share $X_1^* = g^{ek_1} \cdot N_1^{sk}$, where $g \in \mathcal{G}$ denotes the group generator, sk the password, and ek_1 the client's ephemeral key. The server's reply consists of the server's identity s and key share $X_0^* = g^{ek_0} \cdot N_0^{sk}$, where ek_0 denotes its ephemeral key. The shared secret is computed as $Z = (X_p^* \cdot N_p^{-sk})^{ek_{1-p}}$, where $p = 1$

[8] In fact, the output of *HKDF* is variable length, and the desired output length is a parameter of the function. We will think of this parameter as being fixed.

[9] The hash length is the number of bits output by the negotiated hash function. This is the same hash function used for *HKDF*.

[10] The TLS spec uses the term “traffic secret” rather than “traffic key”.

[11] The public parameters are usually referred to as M and N.

```
spec $\mathbf{SPake2\text{-}AP}_{\mathbb{G}}^{\mathcal{C},\mathcal{S}}$:
1  var $PW$ object, $N_1, N_0$ $\mathbf{elem}_{\mathcal{G}}$
2  op (SETUP): $N_1, N_0 \twoheadleftarrow \mathcal{G}$
3  op (MOVES): ret 2
4  op$^{-,\mathcal{R}}$ (SGEN): $pk \leftarrow [\,]$
5      for $i \in \mathcal{C} \cup \mathcal{S}$ do $pk_i \leftarrow (N_1, N_0)$
6      ret $(pk, PW^{\mathcal{R}}(\,))$
7  op (EGEN, ...): $ek \twoheadleftarrow \mathbb{Z}_{|\mathcal{G}|}$; ret $ek$
8  op (GAME ST, $x, i$ str,
9          $(st, j, K$ str, $X_1^*, X_0^*$ $\mathbf{elem}_{\mathcal{G}}$)):
10     if $st \neq$ done then ret $\perp$
11     if $x =$ sid then ret $(X_1^*, X_0^*)$
12     if $x =$ pid then ret $j$
13     if $x =$ key then ret $K$
14 // Client sends KEX1
15 op (SEND, $c$ $\mathbf{elem}_{\mathcal{C}}$, $sk, ek$ int, $\perp, \perp, \perp$):
16     $X_1^* \leftarrow g^{ek} \cdot N_1^{sk}$
17     ret ((wait, $X_1^*$), $(c, X_1^*)$)
18 // Client on KEX0
19 op$^{-,\mathcal{R}}$ (SEND, $c$ $\mathbf{elem}_{\mathcal{C}}$, $sk, ek$ int, $\perp$,
20         (wait, $X_1^*$ $\mathbf{elem}_{\mathcal{G}}$), ($s$ $\mathbf{elem}_{\mathcal{S}}$, $X_0^*$ $\mathbf{elem}_{\mathcal{G}}$)):
21     $Z \leftarrow (X_0^* \cdot N_0^{-sk})^{ek}$
22     $ikm \leftarrow (c, s, X_1^*, X_0^*, sk, Z)$
23     $K \leftarrow \mathcal{R}_1(ikm)$
24     ret ((done, $s, K, X_1^*, X_0^*$), $\perp$)
25 // Server on KEX1 sends KEX0
26 op$^{-,\mathcal{R}}$ (SEND, $s$ $\mathbf{elem}_{\mathcal{S}}$, $sk$ table, $ek$ int, $\perp, \perp$
27         ($c$ $\mathbf{elem}_{\mathcal{C}}$, $X_1^*$ $\mathbf{elem}_{\mathcal{G}}$)):
28     $X_0^* \leftarrow g^{ek} \cdot N_0^{sk_c}$; $Z \leftarrow (X_1^* \cdot N_1^{-sk_c})^{ek}$
29     $ikm \leftarrow (c, s, X_1^*, X_0^*, sk_c, Z)$
30     $K \leftarrow \mathcal{R}_1(ikm)$
31     ret ((done, $c, K, X_1^*, X_0^*$), $(s, X_0^*)$)
32 op (SEND, ...): ret (fail, $\perp$)  // Invalid message
```

Figure 4.3: Protocol $\mathbf{SPake2\text{-}AP}_{\mathbb{G}}^{\mathcal{C},\mathcal{S}}$, where $\mathbb{G} = (\mathcal{G}, \cdot)$ is a prime-order, cyclic group with generator g and $\mathcal{S}, \mathcal{C} \subseteq \{0,1\}^*$ are finite, disjoint, non-empty sets. Object PW is a symmetric password generator for $\mathcal{S}, \mathcal{C}, \mathcal{P}$ for some dictionary $\mathcal{P} \subseteq \mathbb{Z}_{|\mathcal{G}|}$.

for the server and $p = 0$ for the client. Finally, each party computes the session key as $K = H(ikm)$, where $ikm = (c, s, X_1^*, X_0^*, sk, Z)$, and terminates. The SID is the pair of key shares (X_1^*, X_0^*). The PID is the identity of the peer: s for the client and c for the server.

This simple protocol is formalized by the eCK-protocol $\mathbf{SPake2\text{-}AP}_{\mathbb{G}}^{\mathcal{C},\mathcal{S}}(PW)$ in Figure 4.3 (cf. [1, Figure 1]). Sets $\mathcal{C}$ and $\mathcal{S}$ are finite, disjoint, non-empty sets of strings denoting the clients and servers respectively. The first key-exchange message, which we call KEX1, is produced by the operator on lines 4.3:15-17 that specifies client c's response to a SEND-query with (reading the last three inputs from left to right) the session input $\perp$, state $\perp$, and inbound message $\perp$. This moves c into the wait state (4.3:17) and outputs KEX1. This message is consumed by the operator on lines 4.3:26-31, which specifies server s's response to a SEND-query with input $\perp$, state $\perp$, and inbound message c, X_1^* (KEX1). This moves s into the done state and outputs KEX0. This message is consumed by the operator on line 4.3:19-24, which specifies client c's response to a SEND-query on input $\perp$, in the wait state, and on inbound message s, X_0^* (KEX0).

Receipt of an invalid message, i.e., any SEND-query other than a properly formatted KEX message, causes the session to move to the fail state, as shown on line 4.3:32. Note that symbol "..." in the operator's pattern is treated like a wildcard so that (SEND, ...) matches any SEND-query that did not match any of the preceding operators.

Key derivation is carried out by a call to $\mathcal{R}_1$. To obtain the concrete protocol, one would use the hash function H to instantiate the first resource in the experiment. However, since all existing analyses model H as an RO [4, 20, 1], we will also use an RO. (See Theorem 3 below.)

The protocol is parameterized by an object PW used to generate the static keys. Syntactically, we require that PW halts and outputs a table sk for which $sk[s][c] = sk[c] \in \mathcal{P}$ for all $(c, s) \in \mathcal{C} \times \mathcal{S}$ and some set $\mathcal{P} \subseteq \mathbb{Z}_{|\mathcal{G}|}$, called the *dictionary*. We refer to such an object as a *symmetric password generator for* $\mathcal{C}, \mathcal{S}, \mathcal{P}$. Following Bellare et al. [27], each client c is in possession of a single password $sk[c] \in \mathcal{P}$, used to authenticate to each server; and each server s is in possession of a table $sk[s]$ that stores the password $sk[s][c]$ shared with each client c. Generally speaking—and for SPAKE2 in particular [4, 20, 1]—passwords are assumed to be uniformly and independently distributed over the dictionary $\mathcal{P}$. In this dissertation, we will say that such a generator is *uniform*.[12]

[12] The password generator models the selection and distribution of passwords among the parties in the protocol. The assumption that passwords are sampled uniformly and independently from some dictionary is widely regarded as overly optimistic (regardless of the dictionary's size). For this reason, the UC framework [53] has emerged as the de facto setting for studying PAKEs [56], since the ideally functionality can be defined for any password distribution. Nevertheless, the concrete security of UC-secure PAKEs against dictionary attacks [27] has, to the best of our knowledge, only been studied under this optimistic assumption (cf. [56, §A]). We suspect, however, that security can be proven under weaker assumptions about the password

Requirements for SPAKE2 Standardization. Although SPAKE2 is being considered for adoption as a standalone protocol standard, it is clear from existing drafts [16, 98] and discussions on the CFRG mailing list [44, 144] that the protocol would not be adopted without modification. In our treatment, we will address what we view to be the two most important changes. The first is to remove the need for trusted setup. Rather than rely on N_1, N_0 being generated in a trustworthy manner, the preferred approach [144] is to pick distinct constants $\mathit{const}_1, \mathit{const}_0 \in \{0,1\}^*$ and compute the parameters as $N_1 \leftarrow H_{\mathbb{G}}(\mathit{const}_1)$ and $N_0 \leftarrow H_{\mathbb{G}}(\mathit{const}_0)$, where $H_{\mathbb{G}} : \{0,1\}^* \to \mathcal{G}$ is a hash function suitable for the given group $\mathbb{G}$ (e.g., a suitable "hash-to-curve" algorithm [73]). The second is that, when SPAKE2 is used with key confirmation—either in the stand alone protocol or embedded in TLS—the protocol should provide an option for agreement on authenticated associated data [98].

4.2.3 Securely Instantiating draft-barnes-tls-pake with SPAKE2

Refer to $\mathbf{SPake2\text{-}TLS}_{\mathbb{G}}^{\mathcal{C},\mathcal{S}}(PW, \mathit{const}_1, \mathit{const}_0)$ specified in Figure 4.4. This object partially specifies the usage of SPAKE2 in TLS. We say "partially" because most of the details of TLS are provided by calls to interface oracle $\mathcal{A}$, which are answered by the adversary's auxiliary interface in the real experiment. Calls to $\mathcal{R}_1$ and $\mathcal{R}_2$ are answered by, respectively, an RO for *HKDF* and an RO for $H_{\mathbb{G}}$. Before being passed to *HKDF*, the input is first encoded using an object *en* with the following properties.

Definition 18 (Encoders and represented sets)**.** A *represented set* is a computable set $\mathcal{X}$ for which $\perp \notin \mathcal{X}$ (cf. "represented groups" in [2, §2.1]). Let $\mathcal{X}$ be a represented set. An *$\mathcal{X}$-encoder* is a functional, halting object *en* that exports the following operators:

- $(1, x\ \mathbf{elem}_{\mathcal{X}})$-$(M\ \mathbf{str})$: the encoding algorithm, returns the encoding M of x as a string.
- $(0, M\ \mathbf{str})$-$(x\ \mathbf{elem}_{\mathcal{X}\cup\{\perp\}})$: the decoding algorithm, returns the element x of $\mathcal{X}$ encoded by string M (or $\perp$ if M does not encode an element of $\mathcal{X}$).

Correctness requires that $en_0(en_1(x)) = x$ for every $x \in \mathcal{X}$. □

The Hellos carry the SPAKE2 key-exchange messages. The first is encoded by the client on line 4.4:22 and decoded by the server on line 39, and the second is encoded by the server on line 42 and decoded by the client on line 28. Value *ikm* (the input to H in SPAKE2) is passed to procedure **KDF** (54-65), which is used to derive the traffic and session keys. Oracle $\mathcal{A}$ (which points to the adversary's aux. interface in the security experiment) chooses the salt and information strings, subject to the constraint that the information strings are distinct.

We refer to the ClientHello as HELLO1 and to the ServerHello as HELLO0. Our spec lumps all other handshake messages into two: AUTH0 for the server's parameter and authentication messages (EncryptedExtensions...Finished); and AUTH1 for the client's authentication messages (Certificate...Finished). This consolidates all traffic-key dependent computations into four $\mathcal{A}$-queries: AUTH0 is computed on line 4.4:45 and verified on line 32 and AUTH1 is computed on line 33 and verified on line 50. This strategy allows us to accurately capture the details of TLS without exhaustively specifying them.

Design Considerations. In the security analysis (cf. Theorem 3) we assume the adversary is non-degenerate, meaning it is specified as $\mathbf{NoDeg}(M, SD)$ for some object M and some functional object SD (see Def. 4). Hence, each $\mathcal{A}$-query is answered deterministically by SD and without carrying state between calls. This restriction turns out to be crucial for keeping the security bound tight: in particular, we need it to be the case that KEX1 and KEX0 can be correctly decoded. (Hence the check that decoding succeeds on lines 4.4:22 and 42; refer to the proof of Theorem 3 for details.) As a consequence, we need to be explicit about the use of randomness in order to realistically capture the details of TLS. Specifically, we modify the ephemeral key generator (EGEN) so that it produces the amount of randomness required by the TLS handshake (as determined by SD; see line 4.4:9).

Recall that draft-barnes-tls-pake requires the PAKE to specify the inputs *psk* and *dhe* to the key schedule, and that *psk* is used to derive the *salt* via the first call to *HKDF*. The *salt* is computed by an $\mathcal{A}$-query (4.4:58), meaning we do not much care how it is chosen. Note that SD is given oracle access to the resources in the experiment (Def. 4), so the auxiliary algorithm may compute *salt* according to the TLS spec.

distribution, say, using the distribution's (conditional) min-entropy.

```
spec $\mathbf{SPake2\text{-}TLS}_{\mathbb{G}}^{\mathcal{C},\mathcal{S}}$:  // $\mathcal{A}$ points to $A_2$ (via a game); $\mathcal{R}$ to resources
var $PW, en$ $\mathbf{object}$, $const_1, const_0$ $\mathbf{str}$
op (MOVES): ret 3
op$^{-,\mathcal{R}}$ (SGEN): $pk \leftarrow [\,]$
  $N_1 \leftarrow \mathcal{R}_2(const_1)$; $N_0 \leftarrow \mathcal{R}_2(const_0)$
  for $i \in \mathcal{C} \cup \mathcal{S}$ do $pk_i \leftarrow (N_1, N_0)$
  ret $(pk, PW^{\mathcal{R}}(\,))$
op$^{\mathcal{A}}$ (EGEN, $i$ $\mathbf{elem}_{\mathcal{C}\cup\mathcal{S}}$, $a$ $\mathbf{any}$):
  var $\rho$ $\mathbf{elem}_{\mathbb{N}}$, $r$ $\mathbf{str}$
  $\rho \leftarrow \mathcal{A}(\textsf{rnd}, i, a)$; if $\rho \neq \bot$ then $r \twoheadleftarrow \{0,1\}^{\rho}$
  $ek \twoheadleftarrow \mathbb{Z}_{|\mathcal{G}|}$; ret $(ek, r)$
op (GAME ST, $x, i$ $\mathbf{str}$,
    $(st, j, K$ $\mathbf{str}$, $X_1^*, X_0^*$ $\mathbf{elem}_{\mathcal{G}}, \ldots)$):
  if $st \notin \{\textsf{done}, \textsf{s wait}\}$ then ret $\bot$
  if $x = \textsf{sid}$ then ret $(X_1^*, X_0^*)$
  if $x = \textsf{pid}$ then ret $j$
  if $x = \textsf{key}$ then ret $K$

// Client sends HELLO1
op$^{\mathcal{A},\mathcal{R}}$ (SEND, $c$ $\mathbf{elem}_{\mathcal{C}}$, $sk$, $(ek$ $\mathbf{int}$, $r$ $\mathbf{any})$, $a$ $\mathbf{any}$, $\bot, \bot$):
  var $hello_1$ $\mathbf{str}$
  $N_1 \leftarrow \mathcal{R}_2(const_1)$; $X_1^* \leftarrow g^{ek} \cdot N_1^{sk}$
  $hello_1 \leftarrow \mathcal{A}(\textsf{c hello}, r, a, c, X_1^*)$; if $\mathcal{A}(\textsf{c kex}, hello_1) \neq (c, X_1^*)$ then ret $(\textsf{fail}, \mathcal{A}(\textsf{proto err}))$
  ret $((\textsf{c wait}, X_1^*, hello_1), hello_1)$
// Client on (HELLO0, AUTH0) sends AUTH1
op$^{\mathcal{A},\mathcal{R}}$ (SEND, $c$ $\mathbf{elem}_{\mathcal{C}}$, $sk$, $(ek$ $\mathbf{int}$, $r$ $\mathbf{any})$, $a$ $\mathbf{any}$,
    $(\textsf{c wait}, X_1^*$ $\mathbf{elem}_{\mathcal{G}}$, $hello_1$ $\mathbf{str})$, $(hello_0, auth_0$ $\mathbf{str})$):
  var $s$ $\mathbf{elem}_{\mathcal{S}}$, $X_0^*$ $\mathbf{elem}_{\mathcal{G}}$, $auth_1$ $\mathbf{str}$
  $(s, X_0^*) \leftarrow \mathcal{A}(\textsf{s kex}, hello_0)$; if $\bot \in \{s, X_0^*\}$ then ret $(\textsf{fail}, \mathcal{A}(\textsf{proto err}))$
  $N_0 \leftarrow \mathcal{R}_2(const_0)$; $Z \leftarrow (X_0^* \cdot N_0^{-sk})^{ek}$; $ikm \leftarrow (c, s, X_1^*, X_0^*, sk, Z)$
  $tr \leftarrow hello_1 \,\|\, hello_0$; $(K_1, K_0, K) \leftarrow \mathbf{KDF}(ikm, c, s, hello_1, hello_0)$
  if $K_1 = \bot$ then ret $(\textsf{fail}, \mathcal{A}(\textsf{proto err}))$
  if $\mathcal{A}(\textsf{s verify}, K_0, (a, tr), auth_0) \neq 1$ then ret $(\textsf{fail}, \mathcal{A}(\textsf{verify err}))$
  $tr \leftarrow tr \,\|\, auth_0$; $auth_1 \leftarrow \mathcal{A}(\textsf{c auth}, K_1, (a, tr), r)$
  ret $((\textsf{done}, s, K, X_1^*, X_0^*), auth_1)$

// Server on HELLO1 sends (HELLO0, AUTH0)
op$^{\mathcal{A},\mathcal{R}}$ (SEND, $s$ $\mathbf{elem}_{\mathcal{S}}$, $sk$ $\mathbf{table}$, $(ek$ $\mathbf{int}$, $r$ $\mathbf{any})$, $a$ $\mathbf{any}$, $\bot$, $hello_1$ $\mathbf{str}$):
  var $c$ $\mathbf{elem}_{\mathcal{C}}$, $X_1^*$ $\mathbf{elem}_{\mathcal{G}}$, $hello_0, auth_0$ $\mathbf{str}$
  $(c, X_1^*) \leftarrow \mathcal{A}(\textsf{c kex}, hello_1)$; if $\bot \in \{c, X_1^*\}$ then ret $(\textsf{fail}, \mathcal{A}(\textsf{proto err}))$
  $N_1 \leftarrow \mathcal{R}_2(const_1)$; $Z \leftarrow (X_1^* \cdot N_1^{-sk_c})^{ek}$; $ikm \leftarrow (c, s, X_1^*, X_0^*, sk_c, Z)$
  $N_0 \leftarrow \mathcal{R}_2(const_0)$; $X_0^* \leftarrow g^{ek} \cdot N_0^{sk_c}$
  $hello_0 \leftarrow \mathcal{A}(\textsf{s hello}, r, a, s, X_0^*)$; if $\mathcal{A}(\textsf{s kex}, hello_0) \neq (s, X_0^*)$ then ret $(\textsf{fail}, \mathcal{A}(\textsf{proto err}))$
  $tr \leftarrow hello_1 \,\|\, hello_0$; $(K_1, K_0, K) \leftarrow \mathbf{KDF}(ikm, c, s, hello_1, hello_0)$
  if $K_1 = \bot$ then ret $(\textsf{fail}, \mathcal{A}(\textsf{proto err}))$
  $auth_0 \leftarrow \mathcal{A}(\textsf{s auth}, K_0, (a, tr), r)$; $tr \leftarrow tr \,\|\, auth_0$
  ret $((\textsf{s wait}, c, K, X_1^*, X_0^*, K_1, tr), (hello_0, auth_0))$
// Server on AUTH1
op$^{\mathcal{A},\mathcal{R}}$ (SEND, $s$ $\mathbf{elem}_{\mathcal{S}}$, $sk$ $\mathbf{table}$, $(ek$ $\mathbf{int}$, $r$ $\mathbf{any})$, $a$ $\mathbf{any}$,
    $(\textsf{s wait}, c, K$ $\mathbf{str}$, $X_1^*, X_0^*$ $\mathbf{elem}_{\mathcal{G}}$, $K_1, tr$ $\mathbf{str})$, $auth_1$ $\mathbf{str}$):
  if $\mathcal{A}(\textsf{c verify}, K_1, (a, tr), auth_1) \neq 1$ then ret $(\textsf{fail}, \mathcal{A}(\textsf{verify err}))$
  ret $((\textsf{done}, c, K, X_1^*, X_0^*), \bot)$

op$^{\mathcal{A}}$ (SEND, $\ldots$): ret $(\textsf{fail}, \mathcal{A}(\textsf{unexpected message}))$

procedure $\mathbf{KDF}(ikm, c, s, hello_1, hello_0)$:
  var $info_1, info_0, info, salt$ $\mathbf{str}$
  $info_1 \leftarrow \mathcal{A}(\textsf{c hs traffic}, hello_1, hello_0)$
  $info_0 \leftarrow \mathcal{A}(\textsf{s hs traffic}, hello_1, hello_0)$
  $info \;\;\leftarrow \mathcal{A}(\textsf{derived}, \quad c, s)$
  $salt \;\;\leftarrow \mathcal{A}(\textsf{salt}, \qquad c, s)$
  if $|\{info_1, info_0, info\}| \neq 3 \,\vee$
      $\bot \in \{info_1, info_0, info, salt\}$
    then ret $(\bot, \bot, \bot)$
  $K_1 \leftarrow \mathcal{R}_1(salt, en_1(ikm), info_1)$
  $K_0 \leftarrow \mathcal{R}_1(salt, en_1(ikm), info_0)$
  $K \;\;\leftarrow \mathcal{R}_1(salt, en_1(ikm), info)$
  ret $(K_1, K_0, K)$
```

Figure 4.4: Protocol $\mathbf{SPake2\text{-}TLS}_{\mathbb{G}}^{\mathcal{C},\mathcal{S}}$, where PW, $\mathbb{G} = (\mathcal{G}, \cdot)$, g, $\mathcal{C}$, and $\mathcal{S}$ are as defined in Figure 4.3. Object en is a $\big(\{0,1\}^* \times \{0,1\}^* \times \mathcal{G} \times \mathcal{G} \times \mathbb{Z}_{|\mathcal{G}|} \times \mathcal{G}\big)$-encoder, where $\mathcal{G}$ is a represented set (Def. 18).

The value of *dhe* is the same as the input *ikm* to H in the original SPAKE2 protocol (4.3:22). A more natural approach might have been to set *dhe* to $H(ikm)$ rather than *ikm*, i.e., let *dhe* be the session key derived in the original protocol. This design may work, but our approach simplifies the proof, as well as being a bit more efficient.

The session key computed by the server is made available in the game state (via the SESSION␣ST-operator) prior to key confirmation (4.4:13). As we discuss in the proof of Theorem 3, this is required for indifferentiability in the **eCK** environment, and hence is an artifact of the formal model. (Intuitively, there is no reason why waiting to release a session key until later should affect security.)

Fail Closed. Finally, a feature of our instantiation of draft-barnes-tls-pake is that the client and server "fail closed". This means that: (1) if the client does not indicate support, then the server must tear down the session (4.4:39); and (2) if the client indicates support for the extension in its request, but the server does not indicate usage in its response, then the client must tear down the session (4.4:28). In fact, Barnes-Friel [16] allow the protocol to "fail open", meaning the client and server may fallback to the standard authentication mechanism if available. But this makes little sense when using PAKE to hedge against PKI failures, since an adversary in possession of the server's signing key (or one of the signing keys in the certificate chain) can easily downgrade the connection [41] to certificate-only authentication and impersonate the server.

Security. We now derive the concrete security of this usage of SPAKE2. Our analysis is in the *weak corruption model* of Bellare et al. [27], which assumes that only static keys (i.e., passwords) and not ephemeral keys can be revealed to the attacker. This is without loss of generality, as all existing analyses of SPAKE2 assume the same corruption model [4, 20, 1]. Our proof also uses the GDH assumption [109], defined below.

Definition 19 (Predicate ϕ_{wc})**.** Let $\phi_{\mathrm{wc}}(tx) = (\nexists\,\alpha)\; tx_\alpha \sim (2, \textsf{EK}, \ldots)$. □

Definition 20 (The GDH problem)**.** Let $\mathbb{G} = (\mathcal{G}, \cdot)$ be a cyclic group with generator $g \in \mathcal{G}$. A *DDH oracle for* $\mathbb{G}$ is a halting object *DDH* for which $DDH(X, Y, Z) = 1$ holds if and only if $\log_g X \cdot \log_g Y = \log_g Z$ for all $X, Y, Z \in \mathcal{G}$. Define $\mathbf{Adv}^{\mathrm{gdh}}_{\mathbb{G}}(A) := \Pr\big[\, x, y \twoheadleftarrow \mathbb{Z}_{|\mathcal{G}|} : A^{DDH}(g^x, g^y) = g^{xy} \,\big]$ to be the advantage of an adversary A in solving the GDH problem for $\mathbb{G}$. Informally, we say the GDH problem is hard for $\mathbb{G}$ if the advantage of every efficient adversary is small. □

Let $k \geq 0$ be an integer; let $const_1, const_0$ be distinct strings; let $\mathbb{G} = (\mathcal{G}, \cdot)$ be a prime-order cyclic group; let $\mathcal{C}, \mathcal{S} \subseteq \{0,1\}^*$ be finite, disjoint, non-empty sets; let $\mathcal{P} \subseteq \mathbb{Z}_{|\mathcal{G}|}$ be a dictionary; and let PW be a uniform, symmetric password-generator for $\mathcal{C}, \mathcal{S}, \mathcal{P}$. Define $\mathcal{T}$ to be the set $\{0,1\}^* \times \{0,1\}^* \times \mathcal{G} \times \mathcal{G} \times \mathbb{Z}_{|\mathcal{G}|} \times \mathcal{G}$. Let $\mathit{\Pi} = \textbf{SPake2-TLS}^{\mathcal{C},\mathcal{S}}_{\mathbb{G}}(PW, const_1, const_0)$, $\tilde{\mathit{\Pi}} = \textbf{SPake2-AP}^{\mathcal{C},\mathcal{S}}_{\mathbb{G}}(PW)$, $X = \mathbf{eCK}(\mathit{\Pi})$, and $\tilde{X} = \mathbf{eCK}(\tilde{\mathit{\Pi}})$. Let $\psi = \phi_{\mathrm{init}} \wedge \phi_{\mathrm{sess}} \wedge \phi_{\mathrm{wc}}$. The following says that for any game G, the G^ψ-security of X (in the ROM for *HKDF* and $H_{\mathbb{G}}$) follows from the G^ψ-security of $\tilde{X}$ (in the ROM for H) under the GDH assumption.

Theorem 3. *Let F be an RO from $(\{0,1\}^*)^3$ to $\{0,1\}^k$, R be an RO from $\{0,1\}^*$ to $\mathcal{G}$, and H be an RO from $\mathcal{T}$ to $\{0,1\}^k$. Let DDH be a DDH oracle for $\mathbb{G}$. For every game G and t_A-time, n.d. adversary A making q_r resource queries, q_s SEND-queries, and q_e EXEC-queries, there exist an n.d. adversary B and GDH-adversary C such that*

$$\mathbf{Adv}^{G^\psi}_{X/(F,R)}(A) \leq \mathbf{Adv}^{G^\psi}_{\tilde{X}/(H,DDH)}(B) + 2q_e \mathbf{Adv}^{\mathrm{gdh}}_{\mathbb{G}}(C) + \frac{2q_s}{|\mathcal{P}|} + \frac{(q_s + 2q_e)^2}{2|\mathcal{G}|}\,,$$

where: DDH is t_{DDH}-time; B runs in time $O(\hat{T})$ and makes at most q_s SEND-queries, q_e EXEC-queries, $O(\hat{Q})$ DDH-queries, and q_r H-queries; C runs in time $O(\hat{T})$ and makes at most $O(\hat{Q})$ DDH-queries; $\hat{T} = t_A(t_A + q_r \cdot t_{DDH})$; and $\hat{Q} = q_r(q_s + q_e)$.

Before giving the proof, let us say a few words about the bound. The claim is proved by first applying Lemma 1, then applying Lemma 2 so that we can argue security using the ψ-indifferentiability of $X/(F, R)$ from $\tilde{X}/(H, DDH)$. The bound reflects the loss in security that results from using the PAKE to derive the traffic keys. The GDH-advantage term is used to bound the probability that derivation of one of these keys during an honest run of the protocol (via EXEC) coincides with a previous RO query; the $2q_s/|\mathcal{P}|$-term is used to bound the probability of the same event occurring during an active attack (via SEND). The simulator kills a session if the SID ever collides with another session other than the partner, which accounts for the final term.

Proof of Theorem 3. By Lemma 1, for every t_S-time simulator S there exists a $O(t_A)$-time adversary D' and a $O(t_A t_S)$-time adversary B such that

$$\mathbf{Adv}^{G^{\psi}}_{X/(F,R)}(A) \le \mathbf{Adv}^{G^{\psi}}_{\tilde{X}/(H,DDH)}(B) + \mathbf{Adv}^{\text{sr-indiff}^{\psi}}_{W/(F,R),\tilde{W}/(H,DDH)}(D',S)\,. \tag{4.2}$$

By Lemma 2 there exists a $O(t_A)$-time adversary D such that

$$\mathbf{Adv}^{\text{sr-indiff}^{\psi}}_{W/(F,R),\tilde{W}/(H,DDH)}(D',S) \le \mathbf{Adv}^{\text{sr-indiff}^{\psi}}_{X/(F,R),\tilde{X}/(H,DDH)}(D,S)\,. \tag{4.3}$$

In the remainder we exhibit an efficient simulator S and adversary C for which

$$\mathbf{Adv}^{\text{sr-indiff}^{\psi}}_{X/(F,R),\tilde{X}/(H,DDH)}(D,S) \le 2q_e \mathbf{Adv}^{\text{gdh}}_{\mathbb{G}}(C) + \frac{2q_s}{|\mathcal{P}|} + \frac{(q_s + 2q_e)^2}{2|\mathcal{G}|}\,. \tag{4.4}$$

Let $g \in \mathcal{G}$ denote the generator of $\mathbb{G}$ and let $1_{\mathbb{G}} \in \mathcal{G}$ denote the group identity. Let M denote the adversary's main algorithm and let SD denote the adversary's auxiliary algorithm (Def. 4). Let i^* denote the lexicographically first element of $\mathcal{C} \cup \mathcal{S}$. We shall assume that the adversary is ψ-respecting, meaning that when M halts, the transcript is deemed valid by ψ with probability 1. This is without loss of generality because ψ is efficiently computable. As a result, we may assume that the adversary initializes the parties first, the adversary initializes each session once and before sending a message to it, and the adversary does not reveal ephemeral keys.

The simulator is $S = \mathbf{Sim}(SD, const_1, const_0, i^*)$ as specified in Figures 4.5, 4.6, and 4.7. It is is not especially complicated, but there are a number of details to attend to. So, before we derive Eq. (4.4), let us first take a moment to clarify its operation. Recall that the simulator gets access to two oracles in the reference experiment: one for the auxiliary interface of $\tilde{X}$, which is used to execute the reference protocol $\tilde{\Pi}$; and another for resources (H, DDH). Its job is to simulate aux./resource queries for $X/(F,R)$. The central problem it must solve is that the adversary has direct access to the main interface of $\tilde{X}$, which provides it with the game and attack state. Hence, the adversary needs to use its own oracles in a way that ensures the game and attack state are consistent with the adversary's view of the execution. In essence, our strategy is to "embed" a session of the reference protocol into each simulation of the real one.

Answering Aux.-Interface Queries. The simulator answers auxiliary queries as follows. When the adversary (i.e., M) wants to initiate a session (s,i) with input a, the simulator queries $\mathcal{X}(\textsf{INIT}, s, i, \bot)$ and stores $\hat{\alpha}^i_s \leftarrow a$, thereby initiating the same session in the reference environment and storing the initial input for subsequent queries to SD (see line 4.5:12 and the definition of **Init** on line 56.) For each session (s,i) the simulator maintains a variable $\hat{\pi}^i_s$, which will be used to keep state for the simulated session. When M wishes to send a message m to session (s,i), the simulator forwards (s,i,m) to interface **Send** defined in Figure 4.6: this induces at most one SEND-query to reference session (s,i), updates the simulation state $\hat{\pi}^i_s$, and produces simulated output (4.5:14). To simulate a passive attack, the simulator first makes an EXEC-query (4.5:17), then extracts the password-masked key-shares from the execution trace (22) and uses them in a sequence of calls to **Send** to simulate an honest run of the real protocol (23-26).

Next we describe **Send**; refer to Figure 4.6. The caller passes in a bit p and an element Y of $\mathcal{G}$. The bit indicates whether the session is under passive attack ($p = 1$; see lines 4.5:23-26) or active attack ($p = 0$; see line 14). In case of a passive attack, the key shares are provided by the caller (i.e., the key share is equal to input $Y \in \mathcal{G}$); in case of an active attack, the key shares are obtained by making queries to the corresponding session (4.6:4 and 22). Except for simulation of the traffic keys, all computations are just as they are in the real protocol. The adversary's auxiliary interface is used to encode the TLS messages, and because the adversary is non-degenerate, the simulator may use the auxiliary algorithm SD as a sub-routine. Thus, each call to the $\mathcal{A}$ oracle in Figure 4.4 is substituted in Figure 4.5 by a call to procedure **SD**, which forwards the call to SD (see line 4.5:50).

In the real experiment (Figure 4.5), the $\mathcal{A}$ oracle is also called upon at various points in order to decide whether execution has encountered an error and will move to the fail state (see lines 4.4:22, 28, 31, 32, 39, 42, 44, 50, and 53). Whenever a session is in this state, the game state should not be obtainable via the main interface: it should only be available in the done state or by the server when it is waiting for key confirmation (4.4:13). To emulate this behavior, the simulator queries $\mathcal{X}(\textsf{SEND}, s, i, \textsf{kill session})$, which forces the reference session into the fail state, since "kill session" is not a valid protocol message (4.3:32). The call is carried out by **Fail** (4.5:57-58), which is called whenever the simulated session is meant to move into the fail state.

```
spec Sim:  // $\mathcal{X}$ points to $\tilde{X}_2$; $\mathcal{R}$ to resources $(H, DDH)$
1  var $SD$ object, $const_1, const_0, i^*$ str
2  var $\hat{pw}, \hat{r}, \hat{\alpha}, \hat{\pi}, derived, traffic, T, U$ table
3  var $\mathcal{Q}$ set, $N_1, N_0$ elem$_\mathcal{G}$
4  op (SETUP): $SD$(SETUP)
5     $\hat{pw}, \hat{r}, \hat{\alpha}, \hat{\pi}, derived, traffic, T, U \leftarrow [\,]$; $\mathcal{Q} \leftarrow \emptyset$
6
7  // Adversarial interface
8  op$^\mathcal{X}$ (2, PK, $i$ str): ret $\mathcal{X}$(PK, $i$)
9  op$^\mathcal{X}$ (2, SK, $i$ str):
10    if $i \in \mathcal{C}$ then $pw_i \leftarrow \mathcal{X}$(SK, $i$); ret $pw_i$
11    else $sk \leftarrow \mathcal{X}$(SK, $i$); for $j \in \mathcal{C}$ do { $\hat{pw}_j \leftarrow sk_j$ }; ret $sk$
12 op$^{\mathcal{X},\mathcal{R}}$ (2, INIT, $s, i$ str, $a$ any): Setup($s, i$); ret Init($s, i, a$)
13 op$^{\mathcal{X},\mathcal{R}}$ (2, SEND, $s, i$ str, $m$ any): Setup($s, i$)
14    $(\hat{\pi}_s^i, out) \leftarrow$ Send($0, 1_\mathbb{G}, s, i, \hat{r}_s^i, \hat{\alpha}_s^i, \hat{\pi}_s^i, m$); ret $out$
15 op$^{\mathcal{X},\mathcal{R}}$ (2, EXEC, $s_1, i_1, s_0, i_0$ str, $a_1, a_0$ any):
16    Setup($s_1, i_1$); Setup($s_0, i_0$)
17    $tr \leftarrow \mathcal{X}$(EXEC, $s_1, i_1, s_0, i_0, a_1, a_0$)
18    if $i_1 \notin \mathcal{C} \vee i_0 \notin \mathcal{S}$ then
19       $err \leftarrow$ SD(unexpected message)
20       ret $(err, err, err, err)$
21    else
22       $((\_, X_1^*), (\_, X_0^*), \_) \leftarrow tr$; $tr \leftarrow (\,)$
23       $(\pi_1, out) \leftarrow$ Send($1, X_1^*, s_1, i_1, a_1, \perp, \perp$);     $tr \leftarrow tr \,.\, out$
24       $(\pi_0, out) \leftarrow$ Send($1, X_0^*, s_0, i_0, a_0, \perp, out$);  $tr \leftarrow tr \,.\, out$
25       $(\pi_1, out) \leftarrow$ Send($1, X_1^*, s_1, i_1, a_1, \pi_1, out$); $tr \leftarrow tr \,.\, out$
26       $(\pi_0, out) \leftarrow$ Send($1, X_0^*, s_0, i_0, a_0, \pi_0, out$); $tr \leftarrow tr \,.\, out$
27       ret $tr$
28
29 // Resource interface
30 op$^{\mathcal{X},\mathcal{R}}$ (3, (1, ($salt, encoded, info$ str))):
31    Setup($\perp, i^*$); ret R($salt, encoded, info$)
32 op$^{\mathcal{X},\mathcal{R}}$ (3, (2, $const$ str)): Setup($\perp, i^*$); ret R($const$)

interface R:
33 op ($salt, encoded, info$ str):
34    if $en_0(encoded) \neq \perp$ then
35       $(i, j, X_1^*, X_0^*, sk, Z) \leftarrow en_0(encoded)$
36       $A \leftarrow X_1^* \cdot N_1^{-sk}$; $B \leftarrow X_0^* \cdot N_0^{-sk}$
37       if $\mathcal{R}_2(A, B, Z) = 1$ then
38          // If derived key, then use reference RO.
39          if $derived[X_1^*, X_0^*, i, j] = 1$ then
40             ret $\mathcal{R}_1(en_0(encoded))$
41          // If traffic key, then simulate response.
42          $K \leftarrow traffic[X_1^*, X_0^*, i, j, salt, info]$
43          if $K \neq \perp$ then ret $K$
44    if $T[salt, encoded, info] = \perp$ then
45       $T[salt, encoded, info] \twoheadleftarrow \{0,1\}^k$
46    ret $T[salt, encoded, info]$
47 op ($const$ str):
48    if $U_{const} = \perp$ then $U_{const} \twoheadleftarrow \mathcal{G}$
49    ret $U_{const}$

procedure SD($in$):
50 ret $SD^\mathbf{R}(in)$

procedure Setup($s, i$):
51 var $\rho$ elem$_\mathbb{N}$
52 $(N_1, N_0) \leftarrow \mathcal{X}$(PK, $i$)
53 $U_{const_1} \leftarrow N_1$; $U_{const_0} \leftarrow N_0$
54 if $s \neq \perp$ then $\rho \leftarrow \mathcal{A}$(rnd, $i, a$)
55 if $\rho \neq \perp \wedge \hat{r}_s^i = \perp$ then $\hat{r}_s^i \twoheadleftarrow \{0,1\}^\rho$

procedure Init($s, i, a$):
56 $\alpha_s^i \leftarrow a$; ret $\mathcal{X}$(INIT, $s, i, \perp$)

procedure Fail($s, i, err$):
57 $\mathcal{X}$(SEND, $s, i$, kill session)
58 ret (fail, SD($err$))
```

Figure 4.5: Specification of simulator S for proof of Theorem 3. **Send** is defined in Figure 4.6.

Procedure **SimKDF** (Figure 4.7) is called upon to simulate computation of the handshake traffic keys K_1 and K_0. Its job is to pick these in a way that provides the adversary with a consistent view of the simulation of RO F. Given only the key shares, the simulator must determine whether the adversary already "knows" what the traffic keys should be. The difficulty is that there are conditions under which the shared secret might be known to the adversary, but is hard to compute for the simulator. Our solution is to distinguish between queries for which the adversary definitely knows what the keys ought to be and queries for which the adversary is "unlikely" to know the keys. In the former case, we can arrange the simulation so that the correct keys are always output. In the latter, we will guess that the adversary does not know the correct output, generate fresh keys, and provide a consistent simulation of F going forward. This back-patching strategy might fail to provide a consistent simulation, so we will need to argue that the probability of failure is small.

Answering Resource Queries and Simulating Traffic Keys. Procedure **SimKDF** has the following inputs: a bit r indicating the role of the session ($r = 1$ for client; $r = 0$ for server); the bit p indicating whether session is under passive attack ($p = 1$) or active attack ($p = 0$); the client i and server j; the Hellos $hello_1$ and $hello_0$; and the key shares X_1^* and X_0^* of the client and server respectively. The simulator maintains a set $\mathcal{Q}$ used to check for and exclude SID collisions (declared at 4.5:3, used at 4.7:3-4): if $(r, X_1^*, X_0^*) \in \mathcal{Q}$, then the procedure immediately halts and returns $(\perp, \perp)$; otherwise, it adds (r, X_1^*, X_0^*) to $\mathcal{Q}$ and proceeds. This step ensures independence of traffic keys computed by partnered sessions, which will simplify the analysis. Next, the procedure computes the salt $salt$ and information strings $info_1, info_0, info$ as usual (4.7:6-13). It then sets $derived[X_1^*, X_0^*, i, j] \leftarrow 1$ in a table $derived$, which will be used by the RO simulation.

```
interface Send:
// Client sends HELLO1
op $(p\ \mathbf{bool}, Y\ \mathbf{elem}_{\mathcal{G}}, s\ \mathbf{str}, i\ \mathbf{elem}_{\mathcal{C}}, a\ \mathbf{any}, \perp, \perp)$:
    var $hello_1$ $\mathbf{str}$; $X_1^* \mathbf{elem}_{\mathcal{G}}$
    if $p = 1$ then $X_1^* \leftarrow Y$ else $(\_, X_1^*) \leftarrow \mathcal{X}(\text{SEND}, s, i, \perp)$ // wait
    $hello_1 \leftarrow \mathbf{SD}(\text{c hello}, \hat{r}_s^i, a, i, X_1^*)$; if $\mathbf{SD}(\text{c kex}, hello_1) \neq (i, X_1^*)$ then ret $\mathbf{Fail}(s, i, \text{proto err})$
    ret $((\text{c wait}, X_1^*, hello_1), hello_1)$
// Client on HELLO0, AUTH0 sends AUTH1
op $(p\ \mathbf{bool}, Y\ \mathbf{elem}_{\mathcal{G}}, s\ \mathbf{str}, i\ \mathbf{elem}_{\mathcal{C}}, a\ \mathbf{any}, (\text{c wait}, X_1^*\ \mathbf{elem}_{\mathcal{G}}, hello_1\ \mathbf{str}), (hello_0, auth_0\ \mathbf{str}))$:
    var $j$ $\mathbf{elem}_{\mathcal{S}}$, $X_0^*$ $\mathbf{elem}_{\mathcal{G}}$, $auth_1$ $\mathbf{str}$
    $(j, X_0^*) \leftarrow \mathbf{SD}(\text{s kex}, hello_0)$; if $\perp \in \{j, X_0^*\}$ then ret $\mathbf{Fail}(s, i, \text{proto err})$
    if $p = 0$ then $\mathcal{X}(\text{SEND}, s, i, (j, X_0^*))$ // done
    $tr \leftarrow hello_1 \,\|\, hello_0$; $(K_1, K_0) \leftarrow \mathbf{SimKDF}(1, p, i, j, hello_1, hello_0, X_1^*, X_0^*)$
    if $K_1 = \perp$ then ret $\mathbf{Fail}(s, i, \text{proto err})$
    if $\mathbf{SD}(\text{s verify}, K_0, (a, tr), auth_0) \neq 1$ then ret $\mathbf{Fail}(s, i, \text{verify err})$
    $tr \leftarrow tr \,\|\, auth_0$; $auth_1 \leftarrow \mathbf{SD}(\text{c auth}, K_1, (a, tr), \hat{r}_s^i)$
    ret $(\text{done}, auth_1)$

// Server on HELLO1 sends HELLO0, AUTH0
op $(p\ \mathbf{bool}, Y\ \mathbf{elem}_{\mathcal{G}}, s\ \mathbf{str}, j\ \mathbf{elem}_{\mathcal{S}}, a\ \mathbf{any}, \perp, hello_1\ \mathbf{str})$:
    var $i$ $\mathbf{elem}_{\mathcal{C}}$, $X_1^*, X_0^*$ $\mathbf{elem}_{\mathcal{G}}$, $hello_0, auth_0$ $\mathbf{str}$
    $(i, X_1^*) \leftarrow \mathbf{SD}(\text{c kex}, hello_1)$; if $\perp \in \{i, X_1^*\}$ then ret $\mathbf{Fail}(s, j, \text{proto err})$
    if $p = 1$ then $X_0^* \leftarrow Y$ else $(\_, X_0^*) \leftarrow \mathcal{X}(\text{SEND}, s, j, (i, X_1^*))$ // done
    $hello_0 \leftarrow \mathbf{SD}(\text{s hello}, \hat{r}_s^j, a, j, X_0^*)$; if $\mathbf{SD}(\text{s kex}, hello_0) \neq (j, X_0^*)$ then ret $\mathbf{Fail}(s, j, \text{proto err})$
    $tr \leftarrow hello_1 \,\|\, hello_0$; $(K_1, K_0) \leftarrow \mathbf{SimKDF}(0, p, i, j, hello_1, hello_0, X_1^*, X_0^*)$
    if $K_1 = \perp$ then ret $\mathbf{Fail}(s, j, \text{proto err})$
    $auth_0 \leftarrow \mathbf{SD}(\text{s auth}, K_0, (a, tx), \hat{r}_s^j)$; $tr \leftarrow tr \,\|\, auth_0$
    ret $((\text{s wait}, K_1, tr), (hello_0, auth_0))$
// Server on AUTH1
op $(p\ \mathbf{bool}, Y\ \mathbf{elem}_{\mathcal{G}}, s\ \mathbf{str}, j\ \mathbf{elem}_{\mathcal{S}}, a\ \mathbf{any}, (\text{s wait}, K_1, tr\ \mathbf{str}), auth_1\ \mathbf{str})$:
    if $\mathbf{SD}(\text{c verify}, K_1, (a, tr), auth_1) \neq 1$ then ret $\mathbf{Fail}(s, j, \text{verify err})$
    ret $(\text{done}, \perp)$

op $(p\ \mathbf{bool}, Y\ \mathbf{elem}_{\mathcal{G}}, s, i\ \mathbf{str}, \ldots)$: ret $\mathbf{Fail}(s, i, \text{unexpected message})$
```

Figure 4.6: Specification of simulator S for proof of Theorem 3 (continued from Figure 4.5). Procedure **SimKDF** is defined in Figure 4.7; **SD** and **Fail** are defined in Figure 4.5.

```
procedure $\mathbf{SimKDF}(r, p, i, j, hello_1, hello_0, X_1^*, X_0^*)$:
1  var $K_1, K_0, info_1, info_0, info, salt$ str
2  // Exclude SID collision.
3  if $(r, X_1^*, X_0^*) \in \mathcal{Q}$ then ret $(\bot, \bot)$
4  $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(r, X_1^*, X_0^*)\}$
5  // Compute salt and info string.
6  $info_1 \leftarrow \mathbf{SD}($c hs traffic$, hello_1, hello_0)$
7  $info_0 \leftarrow \mathbf{SD}($s hs traffic$, hello_1, hello_0)$
8  $info \;\leftarrow \mathbf{SD}($derived$,\quad i, j)$
9  $salt \;\leftarrow \mathbf{SD}($salt$,\quad\;\; i, j)$
10 // For key sep., each info string must be distinct.
11 if $|\{info_1, info_0, info\}| \neq 3 \vee$
12       $\bot \in \{info_1, info_0, info, salt\}$ then
13   ret $(\bot, \bot)$
14 // Mark session key as derived.
15 $derived[X_1^*, X_0^*, i, j] \leftarrow 1$
16 // Simulate computation of traffic keys.
17 for $b \in \{0, 1\}$ do
18   // Check if the output is definitely known.
19   if $p = 0 \wedge \hat{pw}_i \neq \bot$ then
20     $A \leftarrow X_1^* \cdot N_1^{-\hat{pw}_i}$; $B \leftarrow X_0^* \cdot N_0^{-\hat{pw}_i}$
21     // The following can be checked in $O(t_{DDH} \cdot q_r)$-time.
22     if $(\exists\, Z \in \mathcal{G})\, \mathcal{R}_2(A, B, Z) = 1 \wedge$
23         $T[salt, en_1(i, j, X_1^*, X_0^*, \hat{pw}_i, Z), info_b] \neq \bot$ then
24       $K_b \leftarrow T[salt, en_1(i, j, X_1^*, X_0^*, \hat{pw}_i, Z), info_b]$
25   // If check fails, guess that the output is unknown.
26   if $K_b = \bot$ then
27     if $traffic[X_1^*, X_0^*, i, j, salt, info_b] = \bot$ then
28       $traffic[X_1^*, X_0^*, i, j, salt, info_b] \twoheadleftarrow \{0, 1\}^k$
29     $K_b \leftarrow traffic[X_1^*, X_0^*, i, j, salt, info_b]$
30 ret $(K_1, K_0)$
```

Figure 4.7: Specification of simulator S for proof of Theorem 3 (continued from Figure 4.6). **SD** is defined in Figure 4.5.

The simulator maintains a table $\hat{pw}$ of passwords that are known to the adversary at any given time (4.5:9-11). To simulate computation of traffic key K_b, we first check if the adversary has definitely observed K_b being output from the RO. In case the client's password is known ($\hat{pw}_i \neq \bot$) and the session is under active attack, we unmask the key shares (4.7:20) to get ephemeral public keys A and B for the client and server respectively. We then consult a table T (used by the RO simulation) to determine if the query corresponds to a previous RO query. We proceed as follows: if there exists $Z \in \mathcal{G}$ such that (A, B, Z) is a DDH triple (i.e., $\mathcal{R}_2(A, B, Z) = 1$) and $K = T[salt, en_1(i, j, X_1^*, X_0^*, \hat{pw}_i, Z), info_b] \neq \bot$, then we let $K_b = K$ (4.7:21-24). Note that this check can be carried out in $O(t_{DDH} \cdot q_r)$-time, since there are at most q_r entries in T, and for each entry we make one query to DDH (via $\mathcal{R}$). If K_b is not defined, then we set $traffic[X_1^*, X_0^*, i, j, salt, info_1]$ to a uniform random element of $\{0, 1\}^k$ (4.7:25-29) and return it.

The RO simulation works as follows. For queries matching pattern $(1, (salt, encoded, info\ \ \mathbf{str}))$, where $en_0(encoded)$ matches $(i, j\ \mathbf{str},\ X_1^*, X_0^*\ \mathbf{elem}_{\mathcal{G}},\ sk\ \mathbf{int}, Z\ \mathbf{elem}_{\mathcal{G}})$, we first check if $(X_1^*, X_0^*, i, j, salt, info)$ corresponds to a previous call to **SimKDF**. To do so, we unmask the key shares by running $A \leftarrow X_1^* \cdot N_1^{-sk}$, and $B \leftarrow X_0^* \cdot N_0^{-sk}$ and check if (A, B, Z) is a DDH triple. If so, then we check if $derived[X_1^*, X_0^*, i, j] = 1$ (4.5:39). If so, then because SD is functional, the query must coincide with the session key for (X_1^*, X_0^*). Because RO H was used to compute this in the reference experiment, we return $\mathcal{R}_1(en_0(encoded))$ to the caller. Next, we check if $K = traffic[X_1^*, X_0^*, i, j, salt, info]$ is defined. If so, then return it.

Finally, if nothing has yet been returned, then we proceed with the RO simulation in the usual way: check if $T[salt, encoded, info]$ is defined; if not, then do $T[salt, encoded, info] \twoheadleftarrow \{0, 1\}^k$; finally, output $T[salt, encoded, info]$. Likewise, resource queries matching $(2, (const\ \mathbf{str}))$ are answered by lazy-evaluating a table U (4.5:47-49). Note that, prior to any other interaction with the reference environment, the simulator programs U so that $U_{const_1} = N_1$ and $U_{const_0} = N_0$, where N_1 and N_0 are the global parameters in the reference experiment. See procedure **Setup** defined on lines 4.5:51-55.

We now proceed with the proof. Eq. (4.4) is derived by a game playing argument in which we exhibit a sequence of experiments, beginning with the real experiment $\mathbf{Real}^{\mathsf{out}^{\psi}}_{X/(F,R)}(D)$ and ending with the reference experiment $\mathbf{Ref}^{\mathsf{out}^{\psi}}_{X/(H,DDH)}(D, S)$, where each experiment is obtained by modifying the code of the previous one. Eq. (4.4) is obtained by upper-bounding the probability that the distribution of the output changes between each experiment and the previous one.

Experiment 0. Define procedure $\mathbf{G}^0_{F,R}$ so that $\mathbf{G}^0_{F,R}(D) = \mathbf{Real}^{\mathsf{out}^{\psi}}_{X/(F,R)}(D)$, but modify the protocol specification so that procedure **KDF** sets a flag $bad_1 \leftarrow 1$ if, at any point in the experiment, any two sessions pertaining to the same role (either client or server) compute the same SID (X_1^*, X_0^*). (Note that is the same condition that leads

SimKDF to output $(\perp, \perp)$ on line 4.7:3.) This change has no affect on the outcome of the experiment, so

$$\Pr\left[\, \mathbf{G}^0_{F,R}(D) \,\right] = \Pr\left[\, \mathbf{Real}^{\mathsf{out}^{\psi}}_{X/(F,R)}(D) \,\right] . \tag{4.5}$$

Revision 0-1. Define procedure $\mathbf{G}^1_{F,R}$ from $\mathbf{G}^0_{F,R}$ by having **KDF** immediately halt and output $(\perp, \perp, \perp)$ after bad_1 gets set. This occurs only if the SIDs of any two sessions collide. One can show that the probability of an SID collision is at most the probability of two sessions having the same ephemeral key. Because these keys are independently and uniformly distributed in $\mathbb{Z}_{|\mathcal{G}|}$, a birthday bound yields

$$\Pr\left[\, \mathbf{G}^0_{F,R}(D) \,\right] - \Pr\left[\, \mathbf{G}^1_{F,R}(D) \,\right] \le \Pr\left[\, \mathbf{G}^1_{F,R}(D)\, \textbf{sets}\, bad_1 \,\right] \le \frac{(q_s + 2q_e)^2}{2|\mathcal{G}|} . \tag{4.6}$$

The factor of 2 in the numerator accounts for the fact that each EXEC-query initiates (and so generates ephemeral keys for) two sessions.

Revision 1-2. In experiment 2 we replace the real resources (F, R) with the reference resources (H, DDH). Define procedure $\mathbf{G}^2_{H,\mathit{DDH}}$ from $\mathbf{G}^1_{F,R}$ by making the following changes. First, modify the protocol so that it declares $T, U, \mathit{derived}, \mathit{traffic}$ **table**. And instead of computing "$K \leftarrow \mathcal{R}_1(\mathit{salt}, \mathit{en}_1(\mathit{ikm}), \mathit{info})$" (4.4:64), procedure **KDF** runs "$K \leftarrow \mathcal{R}_1(\mathit{ikm})$; $(i, j, X_1^*, X_0^*, \_, \_) \leftarrow \mathit{ikm}$; $\mathit{derived}[X_1^*, X_0^*, i, j] \leftarrow 1$". Second, modify the protocol by replacing each remaining call to $\mathcal{R}$ with a call to interface $\mathbf{R}$ as defined by the simulator on lines 4.5:33-49. Third, modify the experiment so that the adversary's resource queries are answered by $\mathbf{R}$, rather than providing it direct access to the resources. Fourth, replace (F, R) with (H, DDH).

Observe that session keys are now being computed by calls to the reference RO H, whereas the traffic keys are computed by making queries to a simulation of the real ROs F, R. The reference RO takes as input an element of $\mathcal{T}$, but the real RO (now being simulated) takes as input an element of $(\{0,1\}^*)^3$. That is, the reference RO only takes in ikm, whereas the real RO takes in the salt salt, $\mathit{encoded} = \mathit{en}_1(\mathit{ikm})$, and an information string (one of info_1, info_0, or info). If two sessions compute the same ikm but distinct salts and/or information strings, then the session keys would have a high probability (close to 1) of being distinct in experiment 1, but would be the same in experiment 2. However, this situation is impossible: by revision 0-1, there exists no (X_1^*, X_0^*) and distinct (i, j), $(i', j') \in \mathcal{C} \times \mathcal{S}$ for which $\mathit{derived}[X_1^*, X_0^*, i, j] = 1$ and $\mathit{derived}[X_1^*, X_0^*, i', j'] = 1$. It follows that

$$\Pr\left[\, \mathbf{G}^1_{F,R}(D) \,\right] = \Pr\left[\, \mathbf{G}^2_{H,\mathit{DDH}}(D) \,\right] . \tag{4.7}$$

Revision 2-3. Define procedure $\mathbf{G}^3_{H,\mathit{DDH}}$ from $\mathbf{G}^2_{H,\mathit{DDH}}$ by modifying the protocol specification and environment as follows. Modify the environment so that it declares $\widehat{pw}$ **table** and modify the SK-operator so that it populates $\widehat{pw}$ as the simulator does (4.5:9-11). Replace procedure **KDF** with a procedure **SimKDF*** that works just like **SimKDF**, except that it takes ikm as input, and we insert the following code between lines 4.7:24 and 25:

```
if $K_b = \perp$ then
    if $T[\mathit{salt}, \mathit{en}_1(\mathit{ikm}), \mathit{info}_b] \ne \perp$ then
        if $p = 1$ then $\mathit{bad}_4 \leftarrow 1$; $K_b \leftarrow T[\mathit{salt}, \mathit{en}_1(\mathit{ikm}), \mathit{info}_b]$
        if $p = 0$ then $\mathit{bad}_5 \leftarrow 1$; $K_b \leftarrow T[\mathit{salt}, \mathit{en}_1(\mathit{ikm}), \mathit{info}_b]$
```

That is, before simulating the traffic keys via the $\mathit{traffic}$ table, procedure **SimKDF*** first consults the RO table T to ensure the simulation does not override a previous RO query. If the query is incident to a passive attack ($p = 1$), then the code sets a flag bad_4; if the query is incident to an active attack ($p = 0$), then the code sets a flag bad_5. Whenever one of these conditions is met, **SimKDF*** sets the traffic key to be consistent with the RO simulation. The substantive change in experiment 3 is that we simulate computation of the traffic keys just as the simulator does, as long as doing so does not result in an inconsistent view of the RO. This change does not affect the adversary's view, so

$$\Pr\left[\, \mathbf{G}^2_{H,\mathit{DDH}}(D) \,\right] = \Pr\left[\, \mathbf{G}^3_{H,\mathit{DDH}}(D) \,\right] . \tag{4.8}$$

Revision 3-4. Define procedure $\mathbf{G}^4_{H,DDH}$ from procedure $\mathbf{G}^3_{H,DDH}$ by removing the code "$K_b \leftarrow T[\mathit{salt}, \mathit{en}_1(\mathit{ikm}), \mathit{info}_b]$" immediately after bad_4 gets set. This occurs if the adversary queried RO query $(\mathit{salt}, (i, j, X_1^*, X_0^*, \mathit{sk}, Z), \mathit{info}_b)$ prior to a passive attack for which the SID is (X_1^*, X_0^*). Because SD is functional (by Def. 4), and because the parties ensure that the Hellos correctly encode the key shares (4.4:22 and 42), the adversary cannot cause the parties to compute the shared secret incorrectly. In particular, both $X = X_1^* \cdot N_1^{-sk}$ and $Y = X_0^* \cdot N_0^{-sk}$ are independently and uniformly distributed elements of $\mathcal{G}$, and $Z = g^{xy}$ for $x = \log_g X$ and $y = \log_g Y$. Hence, an efficient adversary D who sets bad_4 with non-negligible probability implies a way to solve the GDH problem for $\mathbb{G}$.

Define GDH-adversary C as follows. On input of $(\Theta, \Phi\ \mathbf{elem}_{\mathcal{G}})$ and with DDH oracle named $\mathcal{DDH}$, choose $n \twoheadleftarrow [q_e]$ and run $\mathbf{G}^4_{H,\mathcal{DDH}}(D)$. On the n-th EXEC-query to the auxiliary interface, compute the key shares as $X_1^* = \Theta \cdot N_1^{sk}$ and $X_0^* = \Phi \cdot N_0^{sk}$, where sk is the password generated by C as part of its simulation of experiment 4. For the sessions corresponding to SID (X_1^*, X_0^*), we do not know the shared secret, but this is only used in experiment 4 to check if bad_4 or bad_5 should be set. We can disregard bad_5, since it is only set during an active attack; and in experiment 4, nothing happens after bad_4 gets set. Instead of checking if bad_4 should be set, the adversary does as follows. During the n-th EXEC-query, replace $\mathbf{SimKDF}^*$ with $\mathbf{SimKDF}$ as defined in Figure 4.7, except that we insert some code between lines 4.7:24 and 25. The code checks if there exists $Z \in \mathcal{G}$ such that $\mathcal{DDH}(\Theta, \Phi, Z) = 1$ and $T[\mathit{salt}, \mathit{en}_1(i, j, X_1^*, X_0^*, \mathit{sk}_i, Z), \mathit{info}_b] \neq \bot$. If such a Z exists, then C immediately halts and outputs Z. The probability that adversary C wins is at least the probability that bad_4 gets set during the n-th EXEC-query. Because n is uniform in $[q_e]$, and $\mathbf{SimKDF}$ is called twice during an honest run, we conclude that

$$\Pr\left[\, \mathbf{G}^3_{H,DDH}(D) \,\right] - \Pr\left[\, \mathbf{G}^4_{H,DDH}(D) \,\right] \leq \Pr\left[\, \mathbf{G}^4_{H,DDH}(D)\ \mathbf{sets}\ \mathit{bad}_4 \,\right] \leq 2q_e \mathbf{Adv}^{\mathrm{gdh}}_{\mathbb{G}}(C)\,. \tag{4.9}$$

Revision 4-5. Finally, define procedure $\mathbf{G}^5_{H,DDH}$ from $\mathbf{G}^4_{H,DDH}$ by removing the code immediately after bad_5 gets set. This occurs if, prior to some SEND-query, the RO table was set at point $(\mathit{salt}, \mathit{en}_1(i, j, X_1^*, X_0^*, \mathit{sk}, Z), \mathit{info}_b)$ for some $b \in \{0, 1\}$, where (X_1^*, X_0^*) is the SID computed by the session, i and j are the client and server, salt and info_b are computed as in $\mathbf{SimKDF}^*$, and Z is the shared secret. Having reached this point in the code, we know that the check on line 4.7:22-23 failed. This implies that sk was never previously revealed to the adversary ($\hat{pw}_i = \bot$). Because the SID (X_1^*, X_0^*) is unique to the session (and its partner, if it exists), the password sk determines a unique point in the RO table that would be set for a given session (or its partner). Because the password generator is uniform, the probability of bad_5 getting set by any one SEND-query is at most $2/|\mathcal{P}|$. Summing over all such queries yields

$$\Pr\left[\, \mathbf{G}^4_{H,DDH}(D) \,\right] - \Pr\left[\, \mathbf{G}^5_{H,DDH}(D) \,\right] \leq \Pr\left[\, \mathbf{G}^5_{H,DDH}(D)\ \mathbf{sets}\ \mathit{bad}_5 \,\right] \leq \frac{2q_s}{|\mathcal{P}|}\,. \tag{4.10}$$

Experiment 5. Noting that $\mathbf{SimKDF}^*$ as defined in experiment 5 is the same as $\mathbf{SimKDF}$ as defined in Figure 4.7, experiment 5 is functionally equivalent to the reference experiment. We conclude that

$$\Pr\left[\, \mathbf{G}^5_{H,DDH}(D) \,\right] = \Pr\left[\, \mathbf{Ref}^{\mathsf{out}^{\psi}}_{\tilde{X}/(H,DDH)}(D, S) \,\right]. \tag{4.11}$$

Resources. The runtime of S includes the runtime of SD, since it uses this algorithm as a sub-routine. On any given call, it invokes SD at most a constant number of times. Each call to SD results in at most q_r queries to the RO, each of which can be evaluated in time $O(t_{DDH})$. Procedure $\mathbf{SimKDF}$ is called at most once, each call requiring $O(q_r \cdot t_{DDH})$-time to evaluate. Finally, each call to $\mathcal{X}$ or $\mathcal{R}$ can be evaluated in constant time. The runtime of S is therefore $O(t_A + q_r \cdot t_{DDH})$. Each call to its 2-interface results in at most 1 SEND-query, 1 EXEC-query, and $2q_r$ DDH-queries. Each call to its 3-interface results in at most 1 H-query and 1 DDH-query. The runtime of B is therefore $O(t_A(t_A + q_r \cdot t_{DDH}))$, and B makes at most q_s SEND-queries, q_e EXEC-queries, $2q_r(q_s + q_e) + q_r = O(q_r(q_s + q_e))$ DDH-queries, and q_r H-queries.

Adversary C runs M in experiment 4 until a certain condition is meant. For each of M's aux./resource queries, it does the same work as S: in particular, SEND- and EXEC-queries are $O(t_A + q_r \cdot t_{DDH})$-time and resource queries are $O(t_{DDH})$-time. Its runtime is therefore $O((q_s + q_e)(t_A + q_r \cdot t_{DDH})) = O(t_A(t_A + q_r \cdot t_{DDH}))$, since $q_s + q_e \leq t_A$. Finally, it too makes at most $O(q_r(q_s + q_e))$ DDH-queries. ■

Remark 5. Let us briefly comment on the implications of using a DDH oracle in the resources for the reference experiment. Any reduction to the reference experiment (i.e., any assumption used to bound B's advantage) must provide the attacker with this capability. In particular, suppose we find a reduction from the CDH assumption to the G^{ψ}-security of $\tilde{X}$ for some game G. Then the stronger GDH assumption would be required in order to use the reduction to prove G^{ψ}-security of X via Theorem 3. (Likewise for DLP and Gap DLP [65].)

4.3 Discussion

In this chapter we considered the problem of protocol translation, where the real system is obtained from the reference system by changing the protocol's specification. The systems in question are $\mathbf{eCK}(\mathit{\Pi})$ and $\mathbf{eCK}(\tilde{\mathit{\Pi}})$ respectively, where $\mathit{\Pi}$ is some "real" AKE protocol, $\tilde{\mathit{\Pi}}$ is some "reference" AKE protocol, and **eCK** specifies the execution environment for the eCK security model [99]. The translation framework allows us to argue that $\mathit{\Pi}$ is at least as secure as $\tilde{\mathit{\Pi}}$ by proving that the execution of $\mathit{\Pi}$ is indifferentiable from the execution of $\tilde{\mathit{\Pi}}$ (in the environment specified by **eCK**). In particular, our analysis of SPAKE2 shows that its usage in TLS is at least as secure as SPAKE2 itself, with only a modest loss in concrete security. Moreover, our result (Theorem 3) "lifts" all existing game-based treatments of SPAKE2 [4, 20, 1], in the sense that the security models in these works can be captured as games in our framework.

However, there is a another security consideration for our extension that Theorem 3 does not explicitly address. Although we have established that the TLS extension meets its intended security goal (i.e., that of SPAKE2), it remains to be seen whether the extension's availability leads to a cross protocol attack against TLS. This seems unlikely, given that the client and server fail closed if the extension is not negotiated; but our analysis does not offer any formal evidence either way.

In the next chapter, we consider an attack model that might help to shed light on the security impact of extensions. Chapter 5 considers the "environment translation" problem, in which the scheme stays the same but its execution environment changes from the reference system to the real system. In particular, we study the problem of designing cryptographic APIs that securely expose secret keys for use in multiple applications. At the end of the next chapter, we will discuss how the same formal approach could be used to reason about extensibility in complex protocols, such as TLS.

Chapter 5

Environment Translation

The principle of key separation, or ensuring that distinct cryptographic functionalities use distinct keys, is a widely accepted tenet of applied cryptography. It appears to be difficult to follow, however, as there are many instances of key reuse in deployed cryptosystems, some having significant impact on the security of applications. There are a number of practical matters that lead to key reuse. First, operational requirements of the system often demand some degree of it. For example, it is common to use a signing key deployed for TLS [123] in other protocols, as this is permitted by certificate authorities and avoids the cost of certifying a distinct key for each protocol. But doing so has side-effects that must be addressed in the design of these protocols, as well as the API that exposes the key to applications [40]. Second, it is often not clear what constitutes a "distinct functionality". Intel's Trusted Platform Module (TPM) standard [145] supports a suite of protocols for remote attestation that all use the same key stored on chip. The TPM exposes a core set of operations involving this key via an API that applications make calls to in order to implement attestation schemes. But the requirement to support so many protocols has lead to a flexible API with subtle vulnerabilities [6, 52].

Prior work sheds light on when key reuse is safe among specific sets of cryptographic primitives. The notion of joint security [82], first discussed in §2.4.2, captures the security of a target cryptosystem (i.e., a digital signature scheme) in the presence of an oracle that exposes a related secret key operation (i.e., the decryption operation of a public-key encryption scheme). Many widely used primitives are jointly secure, including RSA-PSS/OAEP [82] and Schnorr signatures/hybrid encryption [65], in the ROM. (Joint security is also known to hold for signing/encryption under more general conditions [115, 25], in the standard model.) Acar et al. [5] address the related problem of agility, where the goal is to identify multiple instantiations of a particular primitive (e.g., sets of AEAD schemes, PRFs, or signature schemes) that can securely use the same key material. But the range of potential key-reuse attacks goes well beyond what these works cover: attack vectors sometimes break the intended abstraction boundary of the scheme by exposing lower level operations [51, 6], or involve unforeseen protocol interactions at a higher level of abstraction [88, 40].

The goal of this chapter is to devise a security model that accounts for the full range of key reuse observed in the wild. To this end, we propose to surface the API as a first class security object. For our purposes, the API is the component of a system that exposes to applications a fixed set of operations involving one or more secret keys. APIs are often the root-of-trust of applications: TPM, Intel's Software Guard Extensions (SGX), hardware security modules (HSMs), and even chip-and-pin credit cards all provide cryptographic APIs that aim to be trustworthy-by-design. But pressure to meet operational requirements, while exporting interfaces that are suitable for a variety of applications, often leads to vulnerabilities (cf. §1.4). An analogous situation arises in the development of software that uses a cryptographic library: software engineers tend to trust that any use case permitted by an API is secure, without fully grasping its side-effects [110]; and this can lead to vulnerable code [7, 111].

In the framework of Chapter 2, we elucidate a condition under which a cryptographic API is provably safe for use among multiple applications. Called *context separability*, this property is used to relate the security of a target application in its "reference" environment, where we assume the keys are used only for the target application, to its "real" environment in which keys might be used by other applications. Cross protocol attacks involving these other applications are modeled conservatively by allowing the adversary direct access to the same API used by the target application. We formalize this *exposed interface attack* setting in §5.1.

Our notion of context separability is inspired by a design pattern already present in a variety of cryptographic standards. Loosely, this property refers to a cryptographic binding of secret key operations to the application in which they are used, ensuring that operations performed in the context of one application cannot be used to attack another. We provide a formal treatment of two standards, both of which provide some degree of context separability. The first is the IETF-standardized version of the EdDSA signature scheme [87] that takes as input an explicit context string. The second is the Noise protocol framework developed by Trevor Perrin [120], which specifies a set of Diffie-Hellman protocols in which key derivation depends on a string that uniquely identifies the protocol. In §5.2 and §5.3 respectively, we consider the degree to which these mechanisms ensure secure key reuse.

Related Work. In their provable security treatment of the SSH protocol [103], Bergsma et al. [36] formalize a variant of the ACCE model [85] that addresses the common practice of reusing long-term secrets among many SSH ciphersuites. Their main result is a composition theorem that relates the "multiciphersuite-ACCE" security of a protocol to the ACCE security of each ciphersuite in isolation, but in the presence of an "auxiliary oracle" that allows the adversary to obtain signatures under long-term secret keys used in the experiment (cf. [36, Theorem 2]). Security in the presence of such an oracle is generalized by our notion of security under exposed interface attack.

Concurrent to our work is the formal treatment of domain separation of Bellare et al. [22] that addresses the common practice of instantiating multiple, independent random oracles (ROs) using a single hash function. For example, while a security proof might involve three functions H_1, H_2, and H_3 modeled as independent ROs, the scheme's implementation might instantiate these as $H_1(x) = H(\mathit{str}(1) \,\|\, x)$, $H_2(x) = H(\mathit{str}(2) \,\|\, x)$, and $H_3(x) = H(\mathit{str}(3) \,\|\, x)$, where $\mathit{str}(\cdot)$ encodes its input as a fixed-length string and H is a concrete hash function such as SHA-2. The idea is that prefixing the input of H with $\mathit{str}(i)$ is as good as having three independent ROs. And while this seems "obviously true" here, there are other domain-separation schemes which could lead to an attack (in the ROM for H).

Bellare et al. apply the indifferentiability framework in order to distinguish "safe" schemes from those that might lead to attacks. As we will see, context separability and secure "oracle cloning", as they call it, are achieved in much the same way: just as one prefixes the input to an RO with the input's "domain" (i.e., the integer i), we will prefix the input of an RO with the "context" in which a secret key operation is performed. We will see that domain (i.e., context) separation is necessary in our setting, but proving security under exposed interface attack is more involved.

Preliminary work. This chapter is based on a preliminary work by the author that appears in the proceedings of CRYPTO 2019 [117]. There are major differences between the definitions and results presented here and those that appear in the paper; see Remark 7.

5.1 Context-Separable APIs

For our purposes in this chapter, the API is the component of a system that exposes secret keys for use in cryptographic applications. We formalize APIs as objects (§2.1) that, at a minimum, specify how keys are generated.

Definition 21 (API)**.** An *API* is a halting object $\mathit{\Gamma}$ that exports a stateless, (GEN)-(pk, sk **any**)-operator, which we call the *key generator*. We refer to pk as the public key and to sk as the secret key. For symmetric key cryptosystems, the public key is empty (i.e., $pk = \bot$). □

Given an API $\mathit{\Gamma}$ and a target application, our goal is to measure the application's security when $\mathit{\Gamma}$ is used for additional applications. We specify a real and reference execution environment for $\mathit{\Gamma}$, whose indifferentiability will imply that security holds even under exposed interface attack. Refer to specifications **API** and $\widetilde{\mathbf{API}}$ defined in Figure 5.1. System $\tilde{X} = \widetilde{\mathbf{API}}(\mathit{\Gamma})$ provides access to $\mathit{\Gamma}$ as it is used by the target application. The main interface of $\tilde{X}$ provides the caller (i.e., the application) access to three operators. The first is the INIT-operator, which generates and stores a key pair. The second is the SK-operator, which exposes the API (i.e., $\mathit{\Gamma}_{sk}(\cdot)$) for the secret key sk. Finally, the RO-operator provides the application access to the external resources of the experiment. The auxiliary interface of $\tilde{X}$ makes the public key pk available to the caller (i.e., the adversary). System $X = \mathbf{API}(\mathit{\Gamma})$, also specified in

```
spec $\widetilde{\textbf{API}}$:  // $\mathcal{R}$ points to resources                    $\boxed{\textbf{API}}$
var $\mathit{\Gamma}$ object, $pk, sk$ any
op (SETUP): $\mathit{\Gamma}$(SETUP)
// Main interface (used by the game)
op$_{-}^{,\mathcal{R}}$ (1, INIT): $(pk, sk) \leftarrow \mathit{\Gamma}^{\mathcal{R}}$(GEN); ret $pk$
op$_{-}^{,\mathcal{R}}$ (1, SK, $x$ any): ret $\mathit{\Gamma}^{\mathcal{R}}_{sk}(x)$
op$_{-}^{,\mathcal{R}}$ (1, RO, $x$ any): ret $\mathcal{R}(x)$
// Aux. interface (used by the adversary)
op$_{-}^{,\mathcal{R}}$ (2, PK): ret $pk$
$\boxed{\text{op}_{-}^{,\mathcal{R}} \text{ (2, SK, } x \textbf{ any}\text{): ret } \mathit{\Gamma}^{\mathcal{R}}_{sk}(x)}$
```

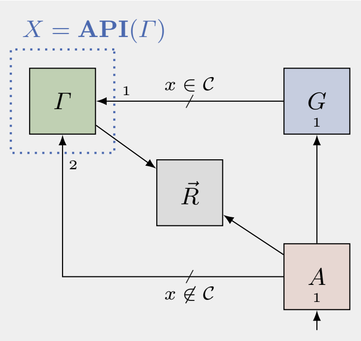

Figure 5.1: Left: Specs **API** and $\widetilde{\textbf{API}}$ used to define context separability. Right: Who may call whom in an exposed-$\mathit{\Gamma}$ attack against G, where $\vec{R}$ denotes the resources of the experiment.

Figure 5.1, is much the same as $\tilde{X}$, except that the auxiliary interface also provides the caller (i.e., the adversary) with access to the API.

Exposed Interface Attacks. Our goal is to devise a formal model for a target application in which the adversary may interact directly with the underlying API. As usual, the security experiment is defined by constructing a world $W = \textbf{Wo}(G, X)$, where G is a game and X is a system (cf. §2.4). In this chapter, the system X will define the execution environment for the API, and the game G will subsume the target application, in addition to the application's intended security goal. This shifts the abstraction boundary relative to Chapters 3 and 4, where the application under attack was subsumed by the system. Here we are less interested in what the application is than how it interacts with the API, hence the shift. In world $W = \textbf{Wo}(G, X)$, where $X = \textbf{API}(\mathit{\Gamma})$, both the adversary A and target application (i.e., the game G) share access to $\mathit{\Gamma}$. We refer to the execution of A in the MAIN experiment for W as an *exposed-$\mathit{\Gamma}$ attack* against (the application codified by) G. Figure 5.1 illustrates which objects have access to which interfaces in this experiment.

This setting conservatively models cross protocol attacks against the target application that result from key reuse. Allowing the adversary unfettered access to the API could lead to attacks that are far more powerful than what is realistic. As an example, suppose the target application codified by G is TLS [123]: if the API provides the adversary with a signing oracle, say for a server, then it is trivial for the adversary to impersonate this server to a client. This is inevitable when the behavior of the "other applications" is unspecified, as it is in our formal model. But in practice we will usually know something about the behavior of these other applications (formalized by the exposed interface attack) that could help to salvage security.

In particular, suppose we know that the set of API calls made by the target application is disjoint from the set of API calls made from all other applications (modeled by the adversary's exposed interface attack). More precisely, for a given game G, suppose there is a set $\mathcal{C}$ such that each of G's queries to the system X is a member of $\mathcal{C}$. In fact, this is true for our TLS example: whenever a signature is produced in TLS, the message that is signed is prefixed by a context string ctx that uniquely identifies the protocol (i.e., TLS 1.3) and the role of the signer (i.e., server). So, if the API is a signing interface for the server, then $\mathcal{C}$ is a subset of $\{msg \in \{0,1\}^* : ctx \preceq msg\}$. This provides a degree of defense against cross protocol attacks (cf. [123, §4.4.3]), since it is unlikely that another application would inadvertently sign messages with the same prefix. If we assume this is the case—that is, suppose that none of the adversary's queries to X coincides with $\mathcal{C}$—then we may be able to prove that TLS is secure against cross protocol attacks involving the server's signing key. (Or the client's, for that matter.)

Context Separability. So that proving security under exposed interface attack is feasible, we shall assume the existence of a set $\mathcal{C}$ that partitions the set of queries made by the game from those made by the adversary. (This is illustrated in Figure 5.1.) In essence, this set formalizes what is known about the real-world behavior of applications that share access to the API: in turn, we will treat it as a parameter of our security experiment that is specific to the API being used. To formalize this, we define a transcript predicate $\hat{\phi}^{\mathcal{C}}_{\text{sep}}$ that deems an attack valid iff each main-interface query is a member of $\mathcal{C}$ and every aux.-interface query is not. This leads immediately to our notion of *context separability*.

Definition 22 (Predicate $\hat{\phi}^{\mathcal{C}}_{\text{sep}}$ and $\mathcal{C}$-separability)**.** Let $\mathcal{C}$ be a computable set and define transcript predicate $\hat{\phi}^{\mathcal{C}}_{\text{sep}}$ so that $\hat{\phi}^{\mathcal{C}}_{\text{sep}}(tx) = 1$ iff $tx_i \sim (1, (op\ \mathbf{str}, in\ \mathbf{any}), \ldots) \Rightarrow (op, in) \in \mathcal{C}$ and $tx_i \sim (2, (\texttt{SK}, in\ \mathbf{any}), \ldots) \Rightarrow (\texttt{SK}, in) \notin \mathcal{C}$ for all $1 \leq i \leq |tx|$. Let Γ be an API, let $X = \mathbf{API}(\Gamma)$, and let $\tilde{X} = \widetilde{\mathbf{API}}(\Gamma)$. Formally, *$\mathcal{C}$-separability* is a measure of the SR-INDIFF$^{\hat{\phi}^{\mathcal{C}}_{\text{sep}}}$ advantage of an adversary in differentiating X from $\tilde{X}$ relative to a simulator (Def. 3). Informally, we say that Γ is $\mathcal{C}$-separable if X is $\hat{\phi}^{\mathcal{C}}_{\text{sep}}$-indifferentiable from $\tilde{X}$; and we say that Γ is context separable if there is a set $\mathcal{C}$ such that $\hat{\phi}^{\mathcal{C}}_{\text{sep}}$ is efficiently computable (Def. 2) and Γ is $\mathcal{C}$-separable. □

Remark 6. The set $\mathcal{C}$ is consists of pairs (op, in), where op denotes one of the system's operators and in denotes the input to the operator. This means that, in addition to $\texttt{SK}$-operator queries, set $\mathcal{C}$ might consist of $\texttt{RO}$-operator queries that are forwarded to the shared resources (5.1:6). □

Our goal is to use an API's context separability to argue that a target application is secure under exposed interface attack. To do so, we will need a variant of the preservation lemma (Lemma 2) for which the class of games is restricted to those whose API queries are disjoint from the adversary's.

Definition 23 (Predicate $\phi^{\mathcal{C}}_{\text{sep}}$ and $\mathcal{C}$-regularity)**.** Let $\mathcal{C}$ be a computable set and define transcript predicate $\phi^{\mathcal{C}}_{\text{sep}}$ so that $\phi^{\mathcal{C}}_{\text{sep}}(tx) = 1$ iff $tx_i \sim (2, (\texttt{SK}, in\ \mathbf{any}), \ldots) \Rightarrow (\texttt{SK}, in) \notin \mathcal{C}$ for all $1 \leq i \leq |tx|$. Let X be a system, G be a game, A be an adversary, and $\vec{R}$ be resources. Run A in the MAIN$^{\phi^{\mathcal{C}}_{\text{sep}}}$ experiment for $\mathbf{Wo}(G, X)/\vec{R}$ and let $\mathcal{Q}$ denote the subset of G's queries to X that match $(op\ \mathbf{str}, in\ \mathbf{any})$. We say that G is *$\mathcal{C}$-regular* if for every such $X, A, \vec{R}$ it holds that $\mathcal{Q} \subseteq \mathcal{C}$ with probability 1. Informally, we say that G is *context regular* if there exists a set $\mathcal{C}$ for which $\phi^{\mathcal{C}}_{\text{sep}}$ is efficiently computable and G is $\mathcal{C}$-regular. □

For context-regular games we have the following lemma which, together with the lifting lemma of Chapter 2, lets us use an API's context separability in order to argue the security of a given application under exposed interface attack.

Lemma 4 (Preservation for regular games)**.** *Let $\mathcal{C}$ be a set, let $\upsilon = \hat{\phi}^{\mathcal{C}}_{\text{sep}}$, and let $\tau = \phi^{\mathcal{C}}_{\text{sep}}$. Let X, Y be systems and $\vec{R}, \vec{Q}$ be resources. For every $(g_1, \_)$-query, $\mathcal{C}$-regular game G, t_A-time, (a_1, a_2, a_r)-query, n.d. adversary A, and simulator S, there exists an n.d. adversary B such that*

$$\mathbf{Adv}^{\text{sr-indiff}^{\tau}}_{W/\vec{R}, V/\vec{Q}}(A, S) \leq \mathbf{Adv}^{\text{sr-indiff}^{\upsilon}}_{X/\vec{R}, Y/\vec{Q}}(B, S)\,,$$

where $W = \mathbf{Wo}(G, X)$, $V = \mathbf{Wo}(G, Y)$, and B is $O(t_A)$-time and $(a_1 g_1, a_2, a_r)$-query.

Proof. We construct adversary B just as we did in Lemma 2. By the $\mathcal{C}$-regularity of G, υ-validity of the transcript holds precisely when τ-validity of the transcript holds. Thus, B's differentiating advantage is precisely the same as A's. ■

Remark 7. Our application of indifferentiability to security under exposed interface attack is comparable to PS19's GAP1 security notion (cf. [117, Def. 5]), but the notions differ in a few important respects. First and foremost, GAP1 is defined in both the standard model and the random oracle model (ROM). In the standard model, GAP1 is a special case of $\phi^{\mathcal{C}}_{\text{sep}}$-indifferentiability of $\mathbf{Wo}(G, \mathbf{API}(\Gamma))$ from $\mathbf{Wo}(G, \widetilde{\mathbf{API}}(\Gamma))$ for a particular set $\mathcal{C}$ and class of APIs Γ and games G. In the ROM, however, their notion is significantly stronger. The relative strength comes from the fact that, in our reference experiment (Figure 2.3), access to the RO (i.e., the resources) is mediated by the simulator; but in the GAP1 experiment, the adversary has direct access to the RO. This affords the simulator less control over the adversary's view of the experiment, resulting in a stronger security property. It is also strictly stronger: as we show in §5.3, we are able to overcome one of their negative results in our setting (cf. [117, Theorem 4]). □

5.2 Case Study: The EdDSA Signature Algorithm

Our first real-world application of context separability is the IETF's standardized version of EdDSA [87], a digital signature scheme originally designed by Bernstein et al. [38]. Unlike more common signatures, like RSA-PSS or ECDSA, the standard admits variants that have an explicit context string, allowing us to easily formalize and prove context separability for them. It specifies two concrete instantiations, whose names indicate the underlying group: Ed25519 and Ed448. Specification $\mathbf{EdDSA}^{b,c,\ell}_{\mathbb{G}}$ in Figure 5.2 defines the generic EdDSA algorithm: a concrete scheme

```
spec $\mathbf{EdDSA}^{b,c,\ell}_{\mathbb{G}}$:   // $\mathcal{R}$ points to resources
var $cl, vr, ph, int, en$ object
op$^{\_,\mathcal{R}}$ (GEN):
    $K \twoheadleftarrow \{0,1\}^b$; $s \leftarrow cl(\mathcal{R}_1(K)[:b])$
    ret $(g^s, (g^s, K))$
op$^{\_,\mathcal{R}}$ (($PK$ elem$_{\mathcal{G}}$, _ str),
            VERIFY, $ctx$, $msg$ str, ($R$ elem$_{\mathcal{G}}$, $x$ int)):
    if $|ctx| > \ell$ then ret $\bot$
    $C \leftarrow vr(ctx)$; $Y \leftarrow ph^{\mathcal{R}}(msg)$
    $t \leftarrow int(\mathcal{R}_1(C \,\|\, en_1(R) \,\|\, en_1(PK) \,\|\, Y))$
    ret $g^{x2^c} = R^{2^c} \cdot PK^{t2^c}$
op$^{\_,\mathcal{R}}$ ((_ elem$_{\mathcal{G}}$, $K$ str),
            SIGN, $ctx$, $msg$ str):
    if $|ctx| > \ell$ then ret $\bot$
    $C \leftarrow vr(ctx)$; $Y \leftarrow ph^{\mathcal{R}}(msg)$
    $W \leftarrow \mathcal{R}_1(K)[b:]$
    $s \leftarrow cl(\mathcal{R}_1(K)[:b])$
    $r \leftarrow int(\mathcal{R}_1(C \,\|\, W \,\|\, Y))$
    $t \leftarrow int(\mathcal{R}_1(C \,\|\, en_1(g^r) \,\|\, en_1(g^s) \,\|\, Y))$
    $x \leftarrow r + st \pmod{|\mathcal{G}|}$
    ret $(g^r, x)$
```

Figure 5.2: Spec $\mathbf{EdDSA}^{b,c,\ell}_{\mathbb{G}}$ for integers $b, c, \ell \geq 0$ and group $\mathbb{G} = (\mathcal{G}, \cdot)$ with generator g, where $|\mathcal{G}| \leq 2^{b-1}$ and $\mathcal{G}$ is a represented set. Object cl computes a function from $\{0,1\}^b$ to $\mathbb{Z}_{|\mathcal{G}|}$; object int computes a bijection from $\{0,1\}^{2b}$ to $\mathbb{Z}_{2^{2b}}$; object en is a $\mathcal{G}$-encoder (Def. 18); object vr computes a function from $\{0,1\}^{\leq \ell}$ to $\{0,1\}^{\ell}$; and ph computes a function from $\{0,1\}^*$ to $\{0,1\}^*$. The first resource in the experiment is an object that computes (an RO for) a function from $\{0,1\}^*$ to $\{0,1\}^{2b}$.

is instantiated by selecting the group $\mathbb{G} = (\mathcal{G}, \cdot)$, integers $b, c, \ell \geq 0$, and functional objects cl, vr, ph, int, and en. The first resource of the experiment (named $\mathcal{R}_1$) should be instantiated with a hash function with range $\{0,1\}^{2b}$.

Let us explain the parameters. The group is determined by a prime number $p > 2$, parameters for a (twisted) Edwards curve E (see [38, Section 2]), and a generator g of a prime order subgroup of $E(\mathbb{F}_p)$, where $E(\mathbb{F}_p)$ denotes the group of points $(x, y) \in \mathbb{F}_p \times \mathbb{F}_p$ that lie on the curve E, and $\mathbb{F}_p$ denotes the finite field of order p. Define b so that $2^{b-1} > p$ and define c so that $\#E(\mathbb{F}_p) = n2^c$ (i.e., 2^c is the cofactor of $\mathbb{G}$). This choice of b makes it possible to encode signatures with $2b$ bits, while the choice of c is intended to mitigate small subgroup attacks [101]. The "clamping" function cl is similarly tailored to the underlying group: for Ed25519 and its variants, this function clears the first 3 bits, sets the second to last bit, and clears the last bit. This ensures that $s = 2^{254} + 8x$ for a uniform random $x \in \mathbb{Z}_{2^{251}}$. Finally, the algorithm variant is determined by the functions vr and ph. For example, the most common Ed25519 variant is obtained by letting $\ell = 0$ so that $vr(ctx) = \varepsilon$ for all ctx and letting $ph(msg) = msg$ for all msg. However, the standard also specifies variants that permit context (Ed25519ctx) and pre-hashing of the message (Ed25519ph). We allow ph to depend on the experiment's resources so that $ph^{\mathcal{R}}(msg)$ can be evaluated as $\mathcal{R}_1(msg)$, as it is for the pre-hashed variants.

Security. EdDSA is a relatively modern signature scheme, designed from the ground-up for speed and to resist a variety of sophisticated attacks. What we aim to prove for EdDSA is relatively simple: that for any string α, its signing API is $\mathcal{C}_\alpha$-separable, where

$$\mathcal{C}_\alpha := \big\{(op, (op', ctx, msg)) \in \{0,1\}^* \times (\{0,1\}^*)^3 : op = \text{SK} \wedge op' = \text{SIGN} \wedge ctx = \alpha\big\}\,. \qquad (5.1)$$

In English, $\mathcal{C}_\alpha$ is the set of signing queries that use α as the context string. We refer to α as the *game context*, as it is the context used exclusively by the game (and is not used by the adversary).

EdDSA specifies an encoding of curve points as bit strings; our argument requires this encoding to be invertible. Namely, object en must be a $\mathcal{G}$-encoder and $\mathcal{G}$ must be represented in the sense of Def. 18. We also require vr to be collision resistant in the following sense.

Definition 24 (CR security)**.** Let F be an object that computes a function. A *collider* for F is a halting object that exports a $(\,)$-$(\mathbf{elem}_{\mathcal{X}}, \mathbf{elem}_{\mathcal{X}})$-operator, where $\mathcal{X}$ is the domain of the function computed by F. Let $\mathbf{Adv}^{\mathrm{cr}}_F(A) := \Pr\big[(x, x') \leftarrow A(\,) : (x \neq x') \wedge (F(x) = F(x'))\big]$ be the CR advantage of collider A in attacking F. $\square$

Theorem 4. *Fix $\mathbb{G} = (\mathcal{G}, \cdot)$, b, c, ℓ, cl, vr, ph, int, en, and $\Gamma = \mathbf{EdDSA}^{b,c,\ell}_{\mathbb{G}}(cl, vr, ph, int, en)$ as specified in Figure 5.2. Fix $\alpha \in \{0,1\}^{\leq \ell}$, let $\mathcal{C} = \mathcal{C}_\alpha$, and let $\tau = \phi^{\mathcal{C}}_{\mathrm{sep}}$. Let $X = \mathbf{API}(\Gamma)$ and $\tilde{X} = \widetilde{\mathbf{API}}(\Gamma)$. Suppose that ph is 1-query and that $|\mathcal{G}| \leq 2^b$. Then for every $(g_1, \_)$-query, $\mathcal{C}$-regular game G and t_A-time, (q_1, q_2, q_r)-query,* *n.d. adversary A there exist an n.d. adversary B and collider C such that*

$$\mathbf{Adv}^{G^\tau}_{X/H}(A) \leq \mathbf{Adv}^{G^\tau}_{\tilde{X}/H}(B) + \mathbf{Adv}^{\mathrm{cr}}_{vr}(C) + \frac{7(q_2)^2 + 5g_1q_1q_r + 3q_2q_r + q_r}{|\mathcal{G}|}\,,$$

where B is $O((t_A)^2)$-time and $(q_1, q_2, q_2{+}q_r)$-query, C is $O(t_A)$-time, and H is an RO from $\{0,1\}^*$ to $\{0,1\}^{2b}$.

Proof. Let $S = \mathbf{Sim}(\,)$ be as specified in Figure 5.3. This object simulates EdDSA signatures by programming the RO much like the simulator for the proof of Theorem 1. Each of its SET-queries (cf. 5.3:8) uses a source (specified on lines 5.3:13-20) that is $(\log|\mathcal{G}|, 2b)$-min-entropy, since x is sampled uniformly from $\mathbb{Z}_{|\mathcal{G}|}$ and T is sampled uniformly from $\{0,1\}^{2b}$. By assumption, each operator defined by G makes at most g_1 queries to X. Each such query results in at most 5 distinct RO queries, since the SIGN-operator makes at most four RO queries and evaluates ph at most once. Hence, an effective query limit of H is $(\hat{Q}{+}q_2, 0)$, where $\hat{Q} = 5g_1q_1{+}5q_2{+}q_r$, since the number of RO queries will not exceed $\hat{q}$. Let P be an RO from $\{0,1\}^*$ to $\{0,1\}^{2b}$ with query limit $(\hat{Q}, q_2)$. By Lemma 1 there exist a $O(t_A)$-time, $O(q_1{+}1, g_2, g_r)$-query n.d. adversary D' and a $O(t_At_S)$-time, $O(q_1, q_2, q_2{+}q_r)$-query n.d. adversary B for which

$$\mathbf{Adv}^{\mathrm{main}^\tau}_{W/H}(A) \leq \mathbf{Adv}^{\mathrm{main}^\tau}_{\tilde{W}/H}(B) + \mathbf{Adv}^{\text{sr-indiff}^\tau}_{W/H,\tilde{W}/P}(D', S) + \frac{q_2(q_2 + \hat{Q})}{|\mathcal{G}|}\,, \tag{5.2}$$

where t_S is the runtime of S, $W = \mathbf{Wo}(G, X)$, and $\tilde{W} = \mathbf{Wo}(G, \tilde{X})$. Let $\upsilon = \hat{\phi}^{\mathcal{C}}_{\mathrm{sep}}$. By Lemma 4 there exists a $O(t_A)$-time, $((q_1{+}1)g_1, q_2, q_r)$-query n.d. adversary D for which

$$\mathbf{Adv}^{\text{sr-indiff}^\tau}_{W/H,\tilde{W}/P}(D', S) \leq \mathbf{Adv}^{\text{sr-indiff}^\upsilon}_{X/H,\tilde{X}/P}(D, S)\,. \tag{5.3}$$

In the remainder we exhibit a $O(t_A)$-time adversary C for which

$$\mathbf{Adv}^{\text{sr-indiff}^\upsilon}_{X/H,\tilde{X}/P}(D, S) \leq \mathbf{Adv}^{\mathrm{cr}}_{vr}(C) + \frac{q_r + 2q_rq_2 + (q_2)^2}{|\mathcal{G}|}\,. \tag{5.4}$$

Combining equations (5.2), (5.3) and (5.4) and simplifying yields the desired bound.

The proof is by a game-playing argument in which we make the following assumptions. The first is that each of D's auxiliary interface queries are distinct. This is without loss because all of the responding operators are deterministic and stateless. The second is that D is υ-respecting, meaning that $\upsilon(tx) = 1$ holds at the end of the experiment. This assumption is without loss because υ is efficiently computable and G is $\mathcal{C}$-regular.

Experiment 0. We begin with a system $X_0 = \mathbf{X}_0(\,)$ whose spec, defined in Figure 5.3, was obtained from the real system X by replacing calls to Γ in the definition of **API** with the responding Γ-operator. For example, the call to Γ's GEN-operator on line 5.1:4 was replaced with the operator defined on lines 5.2:2-4. In addition, the SETUP-operator initializes a random, b-bit string $\overline{W}$. Define procedure $\mathbf{G}^0$ so that $\mathbf{G}^0_H(D) = \mathbf{Real}^{\mathsf{out}^\upsilon}_{X_0/H}(D)$. Since none of these changes affect the semantics of the experiment,

$$\Pr\left[\,\mathbf{Real}^{\mathsf{out}^\upsilon}_{X/H}(D)\,\right] = \Pr\left[\,\mathbf{G}^0_H(D)\,\right]. \tag{5.5}$$

Revision 0-1. Let $X_1 = \mathbf{X}_1(\,)$ as specified in Figure 5.3 and let $\mathbf{G}^1_H(D) = \mathbf{Real}^{\mathsf{out}^\upsilon}_{X_1/H}(D)$. This system is identical to X_0 until a signing query is made via the aux. interface for which $vr(\alpha) = vr(ctx)$, where ctx is the context string incident to the query. If this occurs, then the revised operator immediately halts and returns $\perp$ (5.3:44). We know that $ctx \neq \alpha$ because D is υ-respecting by assumption. Therefore, this condition implies a collision under vr with access to an RO for H. By the fundamental lemma of game playing, we can construct a $O(t_A)$-time collider C for which

$$\Pr\left[\,\mathbf{G}^0_H(D)\,\right] - \Pr\left[\,\mathbf{G}^1_H(D)\,\right] \leq \mathbf{Adv}^{\mathrm{cr}}_{vr}(C)\,. \tag{5.6}$$

The collider simply runs $\mathbf{G}^1_H(D)$ until the condition on line 5.3:44 is met and halts with output (α, ctx). If D halts before this condition is met, then C gives up and outputs $(\varepsilon, \varepsilon)$.

Revision 1-2. The remaining revisions are to the SK-operator exported by the auxiliary interface. Let $X_2 = \mathbf{X}_2(\,)$ as specified in Figure 5.3 and let $\mathbf{G}^2_H(D) = \mathbf{Real}^{\mathsf{out}^\upsilon}_{X_2/H}(D)$. This system changes the way the string W is computed.

```
spec Sim:  // $\mathcal{X}$ points to $\tilde{X}_2$; $\mathcal{R}$ to resources
var $PK$ $\mathbf{elem}_{\mathcal{G}}$
op$^{\mathcal{X},-}$ $(2, \mathsf{PK})$: ret $\mathcal{X}(\mathsf{PK})$
op$^{\mathcal{X},\mathcal{R}}$ $(2, \mathsf{SK}, (\mathsf{SIGN}, ctx, msg\ \mathbf{str}))$:
    $PK \leftarrow \mathcal{X}(\mathsf{PK})$
    if $PK = \bot \vee |ctx| > \ell$ then ret $\bot$
    if $vr(\alpha) = vr(ctx)$ then ret $\bot$
    $C \leftarrow vr(ctx)$; $Y \leftarrow ph^{\mathcal{R}}(msg)$
    $M \leftarrow \mathbf{Src}(\mathcal{X}(\mathsf{PK}), C, Y)$
    $((\_, T), x) \leftarrow \mathcal{R}_1(\mathsf{SET}, M)$; $t \leftarrow int(T)$
    ret $(g^x/PK^t, x)$
op$^{-,\mathcal{R}}$ $(3, (1, \xi\ \mathbf{str}))$:
    ret $\mathcal{R}_1(\xi)$
```

```
spec Src:
var $PK$ $\mathbf{elem}_{\mathcal{G}}$, $C, Y$ $\mathbf{str}$
op $()$:
    $x \twoheadleftarrow \mathbb{Z}_{|\mathcal{G}|}$
    $T \twoheadleftarrow \{0,1\}^{2b}$
    $t \leftarrow int(T)$
    $R \leftarrow g^x/PK^t$
    $\xi \leftarrow C \,\|\, en_1(R) \,\|\, en_1(PK) \,\|\, Y$
    ret $((\xi, T), x)$
```

```
spec $\mathbf{X}_0$:  // $\mathcal{R}$ points to resources $(H, )$          $\boxed{\mathbf{X}_1}$
var $PK$ $\mathbf{elem}_{\mathcal{G}}$, $K, \overline{W}$ $\mathbf{str}$
op $(\mathsf{SETUP})$: $\overline{W} \twoheadleftarrow \{0,1\}^b$
op$^{-,\mathcal{R}}$ $(1, \mathsf{INIT})$:
    $K \twoheadleftarrow \{0,1\}^b$; $s \leftarrow cl(\mathcal{R}_1(K)[:b])$
    $PK \leftarrow g^s$; ret $PK$
op$^{-,\mathcal{R}}$ $(1, \mathsf{RO}, x\ \mathbf{any})$: ret $\mathcal{R}(x)$
op$^{-,\mathcal{R}}$ $(2, \mathsf{PK})$: ret $PK$
op$^{-,\mathcal{R}}$ $(i\ \mathbf{int}, \mathsf{SK}, (\mathsf{VERIFY}, ctx, msg\ \mathbf{str}, (R\ \mathbf{elem}_{\mathcal{G}}, x\ \mathbf{int})))$:
    if $|ctx| > \ell$ then ret $\bot$
    $C \leftarrow vr(ctx)$; $Y \leftarrow ph^{\mathcal{R}}(msg)$
    $t \leftarrow int(\mathcal{R}_1(C \,\|\, en_1(R) \,\|\, en_1(PK) \,\|\, Y))$
    ret $g^{x2^c} = R^{2^c} \cdot PK^{t2^c}$
op$^{-,\mathcal{R}}$ $(1, \mathsf{SK}, (\mathsf{SIGN}, ctx, msg\ \mathbf{str}))$:
    if $|ctx| > \ell$ then ret $\bot$
    $C \leftarrow vr(ctx)$; $Y \leftarrow ph^{\mathcal{R}}(msg)$
    $W \leftarrow \mathcal{R}_1(K)[b:]$
    $r \leftarrow int(\mathcal{R}_1(C \,\|\, W \,\|\, Y))$
    $s \leftarrow cl(\mathcal{R}_1(K)[:b])$
    $t \leftarrow int(\mathcal{R}_1(C \,\|\, en_1(g^r) \,\|\, en_1(g^s) \,\|\, Y))$
    $x \leftarrow r + st \pmod{|\mathcal{G}|}$
    ret $(g^r, x)$
op$^{-,\mathcal{R}}$ $(2, \mathsf{SK}, (\mathsf{SIGN}, ctx, msg\ \mathbf{str}))$:
    if $|ctx| > \ell$ then ret $\bot$
    $\boxed{\text{if } vr(\alpha) = vr(ctx) \text{ then ret } \bot}$
    $C \leftarrow vr(ctx)$; $Y \leftarrow ph^{\mathcal{R}}(msg)$
    $W \leftarrow \mathcal{R}_1(K)[b:]$
    $r \leftarrow int(\mathcal{R}_1(C \,\|\, W \,\|\, Y))$
    $s \leftarrow cl(\mathcal{R}_1(K)[:b])$
    $t \leftarrow int(\mathcal{R}_1(C \,\|\, en_1(g^r) \,\|\, en_1(g^s) \,\|\, Y))$
    $x \leftarrow r + st \pmod{|\mathcal{G}|}$
    ret $(g^r, x)$
```

```
op$^{-,\mathcal{R}}$ $(2, \mathsf{SK}, (\mathsf{SIGN}, ctx, msg\ \mathbf{str}))$:          $\mathbf{X}_1$ $\boxed{\mathbf{X}_2}$
    if $|ctx| > \ell$ then ret $\bot$
    if $vr(\alpha) = vr(ctx)$ then ret $\bot$
    $C \leftarrow vr(ctx)$; $Y \leftarrow ph^{\mathcal{R}}(msg)$
    $W \leftarrow \mathcal{R}_1(K)[b:]$ $\boxed{W \leftarrow \overline{W}}$
    $r \leftarrow int(\mathcal{R}_1(C \,\|\, W \,\|\, Y))$
    $s \leftarrow cl(\mathcal{R}_1(K)[:b])$
    $t \leftarrow int(\mathcal{R}_1(C \,\|\, en_1(g^r) \,\|\, en_1(g^s) \,\|\, Y))$
    $x \leftarrow r + st \pmod{|\mathcal{G}|}$
    ret $(g^r, x)$
```

```
op$^{-,\mathcal{R}}$ $(2, \mathsf{SK}, (\mathsf{SIGN}, ctx, msg\ \mathbf{str}))$:          $\mathbf{X}_2$ $\boxed{\mathbf{X}_3}$
    if $|ctx| > \ell$ then ret $\bot$
    if $vr(\alpha) = vr(ctx)$ then ret $\bot$
    $C \leftarrow vr(ctx)$; $Y \leftarrow ph^{\mathcal{R}}(msg)$
    $W \leftarrow \overline{W}$; $r \leftarrow int(\mathcal{R}_1(C \,\|\, W \,\|\, Y))$
    $\boxed{r \twoheadleftarrow \mathbb{Z}_{2^{2b}}}$; $s \leftarrow cl(\mathcal{R}_1(K)[:b])$
    $t \leftarrow int(\mathcal{R}_1(C \,\|\, en_1(g^r) \,\|\, en_1(g^s) \,\|\, Y))$
    $x \leftarrow r + st \pmod{|\mathcal{G}|}$
    ret $(g^r, x)$
```

```
op$^{-,\mathcal{R}}$ $(2, \mathsf{SK}, (\mathsf{SIGN}, ctx, msg\ \mathbf{str}))$:          $\mathbf{X}_3$ $\boxed{\mathbf{X}_4}$
    if $|ctx| > \ell$ then ret $\bot$
    if $vr(\alpha) = vr(ctx)$ then ret $\bot$
    $C \leftarrow vr(ctx)$; $Y \leftarrow ph^{\mathcal{R}}(msg)$
      $r \twoheadleftarrow \mathbb{Z}_{2^{2b}}$
      $s \leftarrow cl(\mathcal{R}_1(K)[:b])$
      $t \leftarrow int(\mathcal{R}_1(C \,\|\, en_1(g^r) \,\|\, en_1(g^s) \,\|\, Y))$
      $x \leftarrow r + st \pmod{|\mathcal{G}|}$ ret $(g^r, x)$
      $M \leftarrow \mathbf{Src}(PK, C, Y)$
      $(\_, (R, x)) \leftarrow \mathcal{R}_1(\mathsf{SET}, M)$; ret $(R, x)$
```

Figure 5.3: Specifications for the proof of Theorem 4. Top: Specs **Sim** and **Src**. Bottom: Specs $\mathbf{X}_0$, $\mathbf{X}_1$, $\mathbf{X}_2$, $\mathbf{X}_3$, and $\mathbf{X}_4$.

This *randomizer*, as we refer to it in the remainder, is used to generate the integer r used in the signature: in X_1, the randomizer is generated by an RO query on input K; in X_2, the randomizer is instead set to the value of $\overline{W}$ generated during setup. The consequence of this revision is that the randomizer used by the main interface and the auxiliary interface are independently and identically distributed and therefore distinct with high probability. But by revision 0-1, the ℓ-bit prefix C of the RO query used by the auxiliary interface to generate r is distinct from that of the main interface. Therefore, as long as the randomizer used by the main interface is unknown to the adversary, this revision does not change the distribution of signatures computed by the auxiliary interface. This is true as long as the adversary does not correctly "guess" the input K to the RO used to derive the randomizer experiment 1. In particular, it can be shown that

$$\Pr\left[\, \mathbf{G}^1_H(D) \,\right] - \Pr\left[\, \mathbf{G}^2_H(D) \,\right] \leq q_r/2^b \,, \tag{5.7}$$

since K is a uniform random, b-bit string.

Revision 2-3. Let $X_3 = \mathbf{X}_3(\,)$ as specified in Figure 5.3 and let $\mathbf{G}^3_H(D) = \mathbf{Real}^{\mathsf{out}^{\upsilon}}_{X_3/H}(D)$. This revision changes the manner in which r is computed: in X_2, the value of r is the output of an RO query on input of the encoded context C, the randomizer $\overline{W}$, and the pre-hashed message Y; in X_3, the value of r is chosen uniformly from $\mathbb{Z}_{2^{2b}}$. Each query to the auxiliary interface matching (SK, (SIGN, _, _ **str**)) is distinct by assumption. Moreover, since *int* is bijective, the randomizer has the same distribution in both systems. Hence, the distribution of X_3's outputs is identical to the distribution of X_2's outputs as long as none of D's RO queries coincides with a SIGN-operator query to its auxiliary interface. Since $\overline{W}$ is a uniform random, b-bit string, we have

$$\Pr\left[\, \mathbf{G}^2_H(D) \,\right] - \Pr\left[\, \mathbf{G}^3_H(D) \,\right] \leq q_r q_2/2^b \,. \tag{5.8}$$

Revision 3-4. Let $X_4 = \mathbf{X}_4(\,)$ as specified in Figure 5.3 and let $\mathbf{G}^4_P(D) = \mathbf{Real}^{\mathsf{out}^{\upsilon}}_{X_4/P}(D)$. In X_3, the signature is equal to (g^r, x), where $x = r{+}st$ and $t = \mathcal{R}_1(C \,\|\, en_1(g^r) \,\|\, en_1(g^s) \,\|\, Y)$. In X_4, the signature is (R, x) for $R = g^x/PK^t$, where x is chosen uniformly from $\mathbb{Z}_{|\mathcal{G}|}$ and $t = int(T)$ for uniformly chosen T from from $\{0,1\}^{2b}$; and, via a call to source $M = \mathbf{Src}(PK, C, Y)$, the RO is programmed so that a subsequent RO query on $C \,\|\, en_1(R) \,\|\, en_1(PK) \,\|\, Y$ returns T. The revised system is identical to the original until a query to the SIGN-operator of the auxiliary interface overwrites a previously set point.

Consider the probability that the output of source M matches ((ξ, _ **str**), _ **any**) for some string ξ. Source M outputs $((C \,\|\, en_1(R) \,\|\, en_1(PK) \,\|\, Y, T), x)$ where $R = g^x/PK^t$ and $t = int(T)$ for some $PK \in \mathcal{G}$, $T \in \{0,1\}^{2b}$, and $x \in \mathbb{Z}_{|\mathcal{G}|}$. Because en is an encoder, the function $en_1(\cdot)$ is injective. Since x is chosen uniform randomly, the probability that ξ encodes g^x/PK^t is at most $1/|\mathcal{G}|$. In turn, the probability that a call to the auxiliary interface results in a point in the RO being overwritten is at most $1/|\mathcal{G}|$. By revision 0-1, it is only possible to overwrite points set by a resource query or a previous auxiliary-interface query. Hence, the probability of any one auxiliary interface query overwriting a point in the RO is at most $(q_r + q_2)/|\mathcal{G}|$. Summing over all auxiliary-interface queries yields

$$\Pr\left[\, \mathbf{G}^3_H(D) \,\right] - \Pr\left[\, \mathbf{G}^4_P(D) \,\right] \leq q_2(q_r + q_2)/|\mathcal{G}| \,. \tag{5.9}$$

Experiment 4. The execution of adversary D in the real experiment for X_4/P is identical to the execution of D and simulator S in the reference experiment for $\tilde{X}/P$. It follows that

$$\Pr\left[\, \mathbf{G}^4_P(D) \,\right] = \Pr\left[\, \mathbf{Ref}^{\mathsf{out}^{\upsilon}}_{\tilde{X}/P}(D, S) \,\right] . \tag{5.10}$$

Combining each of the above transitions yields

$$\begin{aligned}
\mathbf{Adv}^{\text{sr-indiff}^{\upsilon}}_{X/H,\tilde{X}/P}(D,S) \quad &\leq \quad \mathbf{Adv}^{\text{cr}}_{vr}(C) + \frac{q_r}{2^b} + \frac{q_r q_2}{2^b} + \frac{q_2(q_r + q_2)}{|\mathcal{G}|} && (5.11)\\
&\leq \quad \mathbf{Adv}^{\text{cr}}_{vr}(C) + \frac{q_r}{|\mathcal{G}|} + \frac{q_r q_2}{|\mathcal{G}|} + \frac{q_2(q_r + q_2)}{|\mathcal{G}|} && (5.12)\\
&= \quad \mathbf{Adv}^{\text{cr}}_{vr}(C) + \frac{q_r + 2q_r q_2 + (q_2)^2}{|\mathcal{G}|} \,, && (5.13)
\end{aligned}$$

where Eq. (5.12) follows from the assumption that $|\mathcal{G}| \leq 2^b$. To complete the proof, note that $t_S = O(t_A)$ because the simulator's runtime is linear in the length of A's auxiliary-interface queries. We conclude that B's runtime is $O((t_A)^2)$. Finally, adversary B is $(q_1, q_2, q_r + q_2)$-query since it makes precisely q_1 main-interface queries, at most q_2 aux.-interface queries, and at most $q_r + q_2$ RO queries. ■

Remark 8. Let us address the assumptions made in Theorem 4. First, note that for Ed25519 and Ed448 and their variants, the order of the group $\mathbb{G}$ is less than 2^b [87]. Second, for the contextualized variants Ed25519ctx and Ed448ctx, the function computed by vr is injective, and so the CR-advantage term is 0. Third, for the pre-hashed variants Ed25519ph and Ed448ph, the object ph makes exactly one query to $\mathcal{R}_1$, but the other variants make no such query. □

5.3 Case Study: Key Reuse Among DH Protocols

Many AKE[1] protocols share the following feature. Each authenticating party is in possession of a static key pair $(PK = g^s, s) \in \mathcal{G} \times \mathbb{Z}_{|\mathcal{G}|}$, where $\mathbb{G} = (\mathcal{G}, \cdot)$ is a finite, cyclic group, and all computations involving s have the form "$H(Q^s, \sigma)$" for some hash function H and string σ. Notable examples of protocols in which this "DH-then-hash" operation appears include WireGuard [70], the original QUIC handshake [79], and OPTLS [93], which was the basis of an early draft of TLS 1.3.[2] Our objective in this section is to rule out cross protocol attacks among this class of AKE schemes.

We are especially motivated by a desire to characterize (in)secure key reuse among Noise protocols [120]. Noise provides a set of rules for processing *handshake patterns*, which define the sequence of interactions between an initiator and responder in a protocol. The processing rules involve three primitives: a DH function, an AEAD scheme (cf. Def. 14), and a hash function. Each message sent or received by a host updates the host's state, which consists of the host's ephemeral and static secret keys, the peer's ephemeral and static public keys, shared state used to derive the symmetric key and associated data, the current symmetric key, and the current nonce. The symmetric key, nonce, and associated data are used to encrypt payloads accompanying each message, providing implicit authentication of a peer via confirmation of knowledge of their static secret.

The Noise processing rules give rise to a large set of possible DH protocols, all of which share the core DH-then-hash operation. (WireGuard is an notable example of a protocol in this set.) The processing rules are designed to make it easy to verify properties of handshake patterns, and considerable effort has gone into their formal analysis [71, 89, 102]. But the study of handshake patterns in isolation does not fully address the complexity of using Noise to build and deploy protocols. We have observed that for protocols used widely in practice (e.g., TLS), it is often necessary for the communicants to negotiate the details of the handshake, including the pattern and the set of primitives (cf. Chapter 1). This is out of scope of the core Noise specification, which aims to be as rigid as possible. As a result, there is an apparent gap between our understanding of the security that Noise provides and how it might be used in practice. One question that arises, which we will address here, is whether it is safe to reuse a single static key among many patterns.

To address this question, we consider how to design a context-separable API suitable for this class of protocols. We begin in §5.3.1 with the simplest possible API, which maps the static key s and an arbitrary $Q \in \mathcal{G}$ to Q^s. Due to a key-recovery attack by Brown and Gallant [51], access to such an API is known to significantly degrade the effective security of level of its applications. We provide more evidence of the risk of this API by showing it is not context separable (Theorem 5).

In light of this result, we consider the context separability of DH-then-hash when the hash function is instantiated with *HKDF* [92], as it is in all of the aforementioned protocols [70, 79, 93, 120]. Recall from Chapter 4 that *HKDF* takes as input a salt string, the initial key material (IKM) string, and an information string used to bind derived keys to the context in which they are used. Modeling *HKDF* as an RO, we prove security under exposed interface attack for applications that use distinct information strings when invoking *HKDF* (Theorem 6). Unfortunately, because Noise does not exhibit this behavior, our analysis does not apply directly to Noise. We elaborate on this point at the end of this section.

[1] Authenticated Key Exchange.

[2] The final version of 1.3 [123] is based instead on signed-DH [94], and the latest draft of the IETF standard for QUIC uses the 1.3 handshake [84]. However, DH-then-hash authentication mechanisms have been considered for standardization as an extension for 1.3 [128, 80].

```
spec $\textbf{Static-DH}_{\mathbb{G}}$:
1  op (GEN): $s \twoheadleftarrow \mathbb{Z}_{|\mathcal{G}|}$; ret $(g^s, s)$
2  op ($s$ int, $Q$ elem$_{\mathcal{G}}$): ret $Q^{\,s}$
```

```
spec $\textbf{DH-HKDF}_{\mathbb{G}}$:  // $\mathcal{R}$ points to resources
3  var $vr, en$ object
4  op (GEN): $s \twoheadleftarrow \mathbb{Z}_{|\mathcal{G}|}$; ret $(g^s, s)$
5  op$^{-,\mathcal{R}}$ ($s$ int, $ctx, salt$ str, $Q$ elem$_{\mathcal{G}}$):
6      ret $\mathcal{R}_1(salt, en_1(Q^s), vr(ctx))$
```

Figure 5.4: Specifications $\textbf{Static-DH}_{\mathbb{G}}$ and $\textbf{DH-HKDF}_{\mathbb{G}}$ for finite, cyclic group $\mathbb{G} = (\mathcal{G}, \cdot)$ with generator g, where $\mathcal{G}$ is a represented set (Def. 18). Object vr computes a function from $\{0,1\}^*$ to $\{0,1\}^*$ and object en is a $\mathcal{G}$-encoder.

5.3.1 Insecurity of Static DH

Let $\mathit{\Gamma} = \textbf{Static-DH}_{\mathbb{G}}(\,)$ as specified in Figure 5.4. The key generator chooses $s \twoheadleftarrow \mathbb{Z}_{|\mathcal{G}|}$ and returns (g^s, s); the second operator (5.4:2) takes as input $Q \in \mathcal{G}$ and simply returns $Q^{\,s}$. With the help of such a "static DH oracle", an algorithm devised by Brown and Gallant [51] significantly reduces the cost of computing discrete logarithms in commonly used groups. Given a point $P = g^s \in \mathcal{G}$ and an oracle that computes $Q^{\,s}$ for a chosen input $Q \in \mathcal{G}$, the algorithm correctly computes s in $O(N^{1/3})$ time, where N is the order of $\mathcal{G}$. This is a significant improvement over the $O(N^{1/2})$ complexity of the best known classical algorithms for solving the discrete log problem [121].

In order achieve this time complexity, their algorithm needs to perform about $N^{1/3}$ queries to the API. Making this number of queries is likely to be infeasible in practice, so one might conclude that choosing a large enough group should render a real-world attack impossible. However, we show that exposing static DH is "risky" in the sense that this API could lead to a cross protocol attack against an application. In particular, if the CDH assumption holds for the given group, then the API cannot be proven context separable.

Definition 25 (The CDH problem)**.** Let $\mathbb{G} = (\mathcal{G}, \cdot)$ be a finite, cyclic group with generator $g \in \mathcal{G}$. Define $\mathbf{Adv}^{\mathrm{cdh}}_{\mathbb{G}}(A) := \Pr\big[\, x, y \twoheadleftarrow \mathbb{Z}_{|\mathcal{G}|} \,:\, A(g^x, g^y) = g^{xy} \,\big]$ to be the advantage of an adversary A in solving the CDH problem for $\mathbb{G}$. Informally, we say the CDH problem is hard for $\mathbb{G}$ if the advantage of every efficient adversary is small. □

Let $X = \mathbf{API}(\mathit{\Gamma})$ and $\tilde{X} = \mathbf{API}(\mathit{\Gamma})$. Theorem 5 states that, if the CDH problem is hard for $\mathbb{G}$, then there exists an efficient differentiator of X from $\tilde{X}$ that gets advantage close to 1. By Proposition 2, this rules out the security of applications under exposed-$\mathit{\Gamma}$ attack, at least in general.

Theorem 5. *Suppose $\mathbb{G}$ has prime order. There exists a $O(1)$-time, $(1,1)$-query adversary D such that for every t_S-time simulator S there exists a $O(t_S)$-time adversary A such that* $\mathbf{Adv}^{\mathrm{indiff}}_{X,\tilde{X}}(D,S) = 1 - \mathbf{Adv}^{\mathrm{cdh}}_{\mathbb{G}}(A)$.

Proof. Adversary D is defined as follows. On input of $(1, \textsf{OUT})$ and with oracles $\mathcal{X}$ and $\mathcal{X}'$ for the system's main and auxiliary interface respectively: run $PK \leftarrow \mathcal{X}(\textsf{INIT})$, $r \twoheadleftarrow \mathbb{Z}_{|\mathcal{G}|}$, and $Z \leftarrow \mathcal{X}'(g^r)$; if $Z^{-r} = PK$, then return 1; otherwise return 0. Since the order of $\mathbb{G}$ is prime, there is a unique multiplicative inverse r^{-1} of $r \pmod{|\mathcal{G}|}$. Hence, in the real experiment we have that $Z^{-r} = (Q^{\,s})^{-r} = ((g^r)^s)^{-r} = g^s = PK$, and so $\Pr\big[\,\mathbf{Real}^{\mathsf{out}}_{X}(D)\,\big] = 1$.

Now consider the probability that $Z^{-r} = PK$ in the reference experiment. From S we can construct a CDH adversary A as follows. On input of (PK, Q) run $S(\textsf{SETUP})$, $Z \leftarrow S_2^{\mathbf{X}}(Q)$, and return Z, where $\mathbf{X}$ simulates aux.-interface queries by answering queries matching $(\textsf{PK})$ with PK. Adversary A perfectly emulates the execution of D and S in the reference experiment. Moreover, A succeeds whenever D outputs 1. We conclude that $\Pr\big[\,\mathbf{Ref}^{\mathsf{out}}_{\tilde{X}}(D,S)\,\big] = \mathbf{Adv}^{\mathrm{cdh}}_{\mathbb{G}}(A)$. ■

Remark 9. Our treatment of static DH focuses on intentionally exposing this functionality in an API, but the same kind of argument applies when this exposure is inadvertent. Indeed, the existence of a static DH oracle in an API can be difficult to recognize, and its impact on security is often quite subtle. For example, the weakness in TPM discussed in §1.4 is the result of a static DH oracle inadvertently exposed by the API [6]. A rigorous analysis of the TPM standard in our attack model would have unearthed this subtlety. More generally, we suggest that the approach developed in this chapter could be used to vet API standards before they are implemented to help uncover such flaws. Though the problem with TPM was obvious in hindsight, it is possible that more flaws lurk in this and other API designs. □

5.3.2 Context Separability of DH-then-HKDF

Fix $\mathbb{G} = (\mathcal{G}, \cdot)$, vr, en, and $\mathit{\Gamma} = \mathbf{DH\text{-}HKDF}_{\mathbb{G}}(vr, en)$ as specified in Figure 5.4. Note that the call to *HKDF* is fulfilled by a call to the first resource in the experiment (5.4:6): to obtain the concrete scheme, one would instantiate this resource with *HKDF*. The API's parameters are a $\mathcal{G}$-encoder en (Def. 18) and a functional object vr. Just as in our analysis of EdDSA, we require the function computed by vr to be collision resistant (Def. 24).

Modeling *HKDF* as an RO, we prove context separability of this operation for applications that use $vr(\alpha)$ as the information string for each evaluation of HKDF, where α uniquely identifies the application context. Formally, for all $\alpha \in \{0,1\}^*$ we define

$$\begin{aligned} \mathcal{C}_\alpha := \; & \big\{(op, (ctx, salt, Q)) \in \{0,1\}^* \times (\{0,1\}^* \times \{0,1\}^* \times \mathcal{G}) : op = \textsf{SK} \wedge ctx = \alpha\big\} \cup \\ & \big\{(op, (salt, ikm, info)) \in \{0,1\}^* \times (\{0,1\}^*)^3 : op = \textsf{RO} \wedge info = vr(\alpha)\big\} \end{aligned} \tag{5.14}$$

so that $\mathcal{C}_\alpha$ denotes the set of main-interface queries whose corresponding evaluation of *HKDF*—whether directly via $\textsf{RO}$ or indirectly via $\textsf{SK}$—uses $vr(\alpha)$ as the info string.

Theorem 6. *Fix integer $k \geq 0$ and string α. Let $\mathcal{C} = \mathcal{C}_\alpha$ and $\tau = \phi^{\mathcal{C}}_{\mathrm{sep}}$. Let $X = \mathbf{API}(\mathit{\Gamma})$, $\tilde{X} = \widetilde{\mathbf{API}}(\mathit{\Gamma})$, F be an RO from $(\{0,1\}^*)^3$ to $\{0,1\}^k$, and DDH be a DDH oracle for $\mathbb{G}$. Suppose that $\mathbb{G}$ has prime order. Then for every $\mathcal{C}$-regular game G and t_A-time, (q_1, q_2, q_r)-query, n.d. adversary A there exists an n.d. adversary B and a collider C such that*

$$\mathbf{Adv}^{G^\tau}_{X/F}(A) \leq \mathbf{Adv}^{G^\tau}_{\tilde{X}/(F,DDH)}(B) + \mathbf{Adv}^{\mathrm{cr}}_{vr}(C)\,,$$

where DDH is t_{DDH}-time, B is $O((t_A)^2 + q_2 q_r t_{DDH})$-time, C is $O(t_A)$-time, and B makes q_1 main-interface queries, at most q_2 aux.-interface queries, at most $q_2{+}q_r$ F-queries, and at most $2q_2q_r$ DDH-queries.

Proof. We prove the claim by appealing to the υ-indifferentiability of X/F from $\tilde{X}/(F, DDH)$, where $\upsilon = \hat{\phi}^{\mathcal{C}}_{\mathrm{sep}}$, and exhibit an efficient simulator for which all efficient adversaries have only small SR-INDIFF$^\upsilon$ advantage. Let $S = \mathbf{Sim}(\,)$ as specified in Figure 5.5. By Lemma 1 there exist adversaries B and D' for which

$$\mathbf{Adv}^{G^\tau}_{X/R}(A) \leq \mathbf{Adv}^{G^\tau}_{\tilde{X}/(R,DDH)}(B) + \mathbf{Adv}^{\text{sr-indiff}^\tau}_{W/R,\tilde{W}/(R,DDH)}(D', S)\,, \tag{5.15}$$

where $W = \mathbf{Wo}(G, X)$, $\tilde{W} = \mathbf{Wo}(G, \tilde{X})$, B is $O(t_A t_S)$-time, D' is (t_A)-time, and t_S is the runtime of S. Moreover, D' is $(q_1{+}1, q_2, q_r)$-query and B makes at most q_1 queries to its main interface, and at most q_2 queries to its aux. interface. (We account for B's resource queries at the end.) By Lemma 4 there exists an adversary D for which

$$\mathbf{Adv}^{\text{sr-indiff}^\tau}_{W/R,\tilde{W}/(R,DDH)}(D', S) \leq \mathbf{Adv}^{\text{sr-indiff}^\upsilon}_{X/R,\tilde{X}/(R,DDH)}(D, S)\,. \tag{5.16}$$

In the remainder, we exhibit a $O(t_A)$-time collider C for which

$$\mathbf{Adv}^{\text{sr-indiff}^\upsilon}_{X/R,\tilde{X}/(R,DDH)}(D, S) \leq \mathbf{Adv}^{\mathrm{cr}}_{vr}(C)\,. \tag{5.17}$$

We assume that D is υ-respecting, as defined in the proof of Theorem 4. (This assumption is without loss because υ is efficiently computable and G is $\mathcal{C}$-regular.)

Refer to S's specification **Sim** in Figure 5.5. S's job is to respond to DH-then-HKDF queries made by D without using the secret key s. To do so, it leverages the fact that it also gets to answer D's RO queries. We keep a track of a pair of sets $\mathcal{V}, \overline{\mathcal{V}} \subseteq \mathcal{G}$, where $\mathcal{V}$ is called the set of "real" RO points and $\overline{\mathcal{V}}$ the set of "simulated" RO points. Points corresponding to the former are answered using the real RO F, while those corresponding to the latter are answered by simulating an RO via a table $\overline{T}$. (This is similar to what we did in Theorem 3.)

Prior to the system being initialized (by a call to the $\textsf{INIT}$-operator of the main interface), each aux.-interface query is answered with $\perp$, since this is the output of the real operator prior the secret key being generated. Thereafter, each aux.-interface query matching $(ctx, salt\ \mathbf{str}, Q\ \mathbf{elem}_{\mathcal{G}})$ is answered as follows. If $vr(ctx) = vr(\alpha)$, then halt and output $\perp$. (This ensures that the output is independent of all main-interface queries: we will argue that the probability of this event is small, assuming D is υ-respecting and vr is CR-secure.) Next, consult the set $\mathcal{V}$ to determine if $Q^{\,s} = Z$ for some previous RO query incident to Z: if so, then use the real RO to compute the output. (This condition is determined by a query to DDH, since $Q^{\,s} = Z$ holds just in case $DDH(PK, Q, Z) = 1$ holds, where

```
spec $\mathbf{Sim}$:  // $\mathcal{X}$ points to $\tilde{X}_2$; $\mathcal{R}$ to resources $(F, DDH)$
  var $\overline{T}$ table, $\mathcal{V}, \overline{\mathcal{V}}$ set
  op (SETUP): $\overline{T} \leftarrow []$; $\overline{\mathcal{V}}, \mathcal{V} \leftarrow \emptyset$
  op$^{\mathcal{X}}$ $(2,$ PK$)$:
    if $PK = \bot$ then $PK \leftarrow \mathcal{X}($PK$)$
    ret $PK$
  op$^{\mathcal{X},\mathcal{R}}$ $(2,$ SK$, (ctx, salt$ str$, Q$ elem$_{\mathcal{G}}))$:
    var $PK$ elem$_{\mathcal{G}}$
    $PK \leftarrow \mathcal{X}($PK$)$
    if $PK = \bot$ then $\bot$  // Not initialized
    if $vr(\alpha) = vr(ctx)$ then ret $\bot$
    for $Z \in \mathcal{V}$ do
      if $\mathcal{R}_2(PK, Q, Z) = 1$ then
        ret $\mathcal{R}_1(salt, en_1(Z), vr(ctx))$
    $\overline{\mathcal{V}} \leftarrow \overline{\mathcal{V}} \cup \{Q\}$
    ret $\overline{\mathbf{R}}(ctx, salt, Q)$
  op$^{\mathcal{X},\mathcal{R}}$ $(3, (1, (salt, ikm, info$ str$)))$:
    var $PK, Z$ elem$_{\mathcal{G}}$
    $PK \leftarrow \mathcal{X}($PK$)$
    $Z \leftarrow en_0(ikm)$
    if $Z = \bot \vee info = vr(\alpha)$ then
      ret $\mathcal{R}_1(salt, ikm, info)$
    for $Q \in \overline{\mathcal{V}}$ do  // $PK = \bot \Rightarrow \overline{\mathcal{V}} = \emptyset$
      if $\mathcal{R}_2(PK, Q, Z) = 1$ then
        ret $\overline{\mathbf{R}}(ctx, salt, Q)$
    $\mathcal{V} \leftarrow \mathcal{V} \cup \{Z\}$
    ret $\mathcal{R}_1(salt, ikm, info)$

procedure $\overline{\mathbf{R}}(ctx, salt, Q)$:
  if $\overline{T}[ctx, salt, Q] = \bot$ then
    $\overline{T}[ctx, salt, Q] \twoheadleftarrow \{0,1\}^k$
  ret $\overline{T}[ctx, salt, Q]$
```

```
procedure $\mathbf{G}_F^0(D)$:                                   $\boxed{\mathbf{G}_F^1}$
  var $T, \overline{T}$ table, $\mathcal{V}, \overline{\mathcal{V}}$ set, $PK$ elem$_{\mathcal{G}}$, $s$ int
  $D($SETUP$)$
  $T, \overline{T} \leftarrow []$; $\mathcal{V}, \overline{\mathcal{V}} \leftarrow \emptyset$
  ret $D_1^{\mathbf{X}_1, \mathbf{X}_2, \mathbf{S}_3}($OUT$)$

interface $\mathbf{X}$:
  op $(1,$ INIT$)$: $s \twoheadleftarrow \mathbb{Z}_{|\mathcal{G}|}$; $PK \leftarrow g^s$; ret $PK$
  op $(1,$ SK$, (ctx, salt$ str$, Q$ elem$_{\mathcal{G}}))$:
    ret $\mathbf{R}_1(salt, en_1(Q^s), vr(ctx))$
  op $(1,$ RO$, (1, (salt, ikm, info$ str$)))$:
    ret $\mathbf{R}_1(salt, ikm, info)$
  op $(2,$ PK$)$: ret $PK$
  op $(2,$ SK$, (ctx, salt$ str$, Q$ elem$_{\mathcal{G}}))$:
    $\boxed{\text{if } vr(\alpha) = vr(ctx) \text{ then ret } \bot}$
    ret $\mathbf{R}_1(salt, en_1(Q^s), vr(ctx))$

interface $\mathbf{R}$:
  op $(1, (salt, ikm, info$ str$))$:
    if $T[salt, ikm, info] = \bot$ then
      $T[salt, ikm, info] \twoheadleftarrow \{0,1\}^k$
    ret $T[salt, ikm, info]$

interface $\mathbf{S}$:
  op $(3, (1, (salt, ikm, info$ str$)))$:
    ret $\mathbf{R}_1(salt, ikm, info)$
```

```
interface $\mathbf{X}$:                                   $\boxed{\mathbf{G}_F^2}$ $\mathbf{G}_F^1$
  op $(1,$ INIT$)$: $s \twoheadleftarrow \mathbb{Z}_{|\mathcal{G}|}$; $PK \leftarrow g^s$; ret $PK$
  op $(1,$ SK$, (ctx, salt$ str$, Q$ elem$_{\mathcal{G}}))$:
    ret $\mathbf{R}_1(salt, en_1(Q^s), vr(ctx))$
  op $(1,$ RO$, (1, (salt, ikm, info$ str$)))$:
    ret $\mathbf{R}_1(salt, ikm, info)$
  op $(2,$ PK$)$: ret $PK$
  op $(2,$ SK$, (ctx, salt$ str$, Q$ elem$_{\mathcal{G}}))$:
    if $vr(\alpha) = vr(ctx)$ then ret $\bot$
    ret $\mathbf{R}_1(salt, en_1(Q^s), vr(ctx))$
    for $Z \in \mathcal{V}$ do
      if $Q^s = Z$ then
        ret $\mathbf{R}_1(salt, en_1(Z), vr(ctx))$
    $\overline{\mathcal{V}} \leftarrow \overline{\mathcal{V}} \cup \{Q\}$
    ret $\overline{\mathbf{R}}(ctx, salt, Q)$

interface $\mathbf{S}$:
  op $(3, (1, (salt, ikm, info$ str$)))$:
    var $Z$ elem$_{\mathcal{G}}$
    $Z \leftarrow en_0(ikm)$
    if $Z = \bot \vee info = vr(\alpha)$ then
      ret $\mathbf{R}_1(salt, ikm, info)$
    for $Q \in \overline{\mathcal{V}}$ do
      if $Q^s = Z$ then
        ret $\overline{\mathbf{R}}(ctx, salt, Q)$
    $\mathcal{V} \leftarrow \mathcal{V} \cup \{Z\}$
    ret $\mathbf{R}_1(salt, ikm, info)$

procedure $\overline{\mathbf{R}}(ctx, salt, Q)$:
  if $\overline{T}[ctx, salt, Q] = \bot$ then $\overline{T}[ctx, salt, Q] \twoheadleftarrow \{0,1\}^k$
  ret $\overline{T}[ctx, salt, Q]$
```

Figure 5.5: Specifications for the proof of Theorem 6. Top: Simulator **Sim**. Bottom: Experiments $\mathbf{G}_0$, $\mathbf{G}_1$, and $\mathbf{G}_2$.

$PK = g^s$.) Finally, if no such Z exists, then we use the simulated RO table $\overline{T}$ to compute the output and add Q to the set of simulated RO points $\overline{\mathcal{V}}$.

Each RO query that might coincide with a simulated API query (i.e., those for which *ikm* encodes an element of $\mathcal{G}$ and $\mathit{info} \neq \mathit{vr}(\alpha)$; cf. 5.5:20-21) is answered as follows. First, consult $\overline{\mathcal{V}}$ to determine if $Q^{\,s} = Z$ for some previous API query incident to Q: if so, then use the simulated RO to compute the output. Finally, if no such Q exists, then use the real RO to compute the output and add Z to the set of real RO points $\mathcal{V}$.

Experiment 0. We fix Eq. (5.17) using a game-playing argument. We begin with the procedure $\mathbf{G}^0_F$ defined in the bottom panel of Figure 5.5, which is equivalent to the real experiment for X/F. Its definition was obtained by replacing the procedures $\mathbf{W}$ and $\mathbf{R}$ (cf. Figure 2.3) with pure interfaces $\mathbf{X}$ and $\mathbf{S}$ that emulate the real system X and RO F respectively. Observe the transcript *tx* (and evaluation of the transcript predicate υ) has been removed, and the output of the experiment is simply the output of D. This simplification does not change the outcome of the experiment since D is υ-respecting. Hence,

$$\Pr\big[\,\mathbf{Real}^{\mathsf{out}^\upsilon}_{X/F}(D)\,\big] = \Pr\big[\,\mathbf{G}^0_F(D)\,\big]\,. \tag{5.18}$$

Revision 0-1. Next, procedure $\mathbf{G}^1_F$ is obtained from $\mathbf{G}^0_F$ by changing behavior of the aux.-interface. If $\mathit{vr}(\alpha) = \mathit{vr}(\mathit{ctx})$, where *ctx* is the input context string, then the operator immediately halts and outputs $\bot$ (5.5:41). Since D is υ-respecting, this event occurs only if D makes an aux.-interface query matching $(\mathsf{SK}, (\mathit{ctx}\ \mathbf{str}, \ldots))$ for which $\mathit{ctx} \neq \alpha$ but $\mathit{vr}(\mathit{ctx}) = \mathit{vr}(\alpha)$. In particular, by the fundamental lemma of game playing, there exists a $O(t_A)$-time collider C for which

$$\Pr\big[\,\mathbf{G}^1_F(D)\,\big] - \Pr\big[\,\mathbf{G}^1_F(D)\,\big] \leq \mathbf{Adv}^{\mathrm{cr}}_{\mathit{vr}}(C)\,. \tag{5.19}$$

The collider simply runs $\mathbf{G}^1_F(D)$ until the condition on line 5.5:41 is met and halts with output (α, ctx). If D halts before this condition is met, then C gives up and outputs $(\varepsilon, \varepsilon)$.

Revision 1-2. Next, we modify $\mathbf{X}$ and $\mathbf{S}$ so that the outputs of, respectively, the aux.-interface and RO are answered just as they are by the simulator S. Since there is no *DDH* oracle in the experiment, the conditions on lines 5.5:12 and 23 are evaluated using the secret key (i.e., "$\mathcal{R}_2(PK, Q, Z) = 1$" is replaced with "$Q^{\,s} = Z$"). By correctness of *en*, and because the order of the group $\mathbb{G}$ is prime, the outputs of the modified interfaces are identically distributed to the outputs of the unmodified interfaces. In particular, we have that

$$\Pr\big[\,\mathbf{G}^2_F(D)\,\big] = \Pr\big[\,\mathbf{G}^1_F(D)\,\big]\,. \tag{5.20}$$

Experiment 2. By replacing "$Q^{\,s} = Z$" with "$\mathcal{R}_2(PK, Q, Z) = 1$" we obtain a procedure that is functionally equivalent to the reference experiment with $\tilde{X}/(F, \mathit{DDH})$ and S. We conclude that

$$\Pr\big[\,\mathbf{Ref}^{\mathsf{out}^\upsilon}_{\tilde{X}/F,\mathit{DDH}}(D, S)\,\big] = \Pr\big[\,\mathbf{G}^2_F(D)\,\big]\,. \tag{5.21}$$

To complete the proof, we comment on the runtime and query complexity of B. Adversary B makes precisely as many main-interface queries as A, at most as many aux.-interface queries as A, and at most $q_r + q_2$ RO queries, since A makes q_r RO queries and each call to S results in at most 1 RO query. Finally, adversary B makes at most $2q_2q_r$ *DDH*-queries because each RO query simulated by S results in at most q_2 *DDH*-queries ($|\overline{\mathcal{V}}| \leq q_2$ by definition) and each API query simulated by S results in at most q_r *DDH*-queries ($|\mathcal{V}| \leq q_r$ by definition).

By Lemma 1 the runtime of B is $O(t_A t_S)$, where t_S is the runtime of S. The runtime of each operator defined by S is dominated by: (1) the lengths the inputs, which are all $O(t_A)$ since the inputs are provided by the adversary; and (2) the for-loops, which are both $O(\hat{q}t_{\mathit{DDH}})$-time, where $\hat{q} = \max\{q_2, q_r\}$. This yields an overestimate of B's runtime, since the *DDH* oracle is evaluated at most $2q_2q_r$ times. In particular, the runtime of B is $O((t_A)^2 + q_2q_rt_{\mathit{DDH}})$. ■

Implications for Noise. Noise uses an empty information string for each *HKDF* evaluation (see [120, §4.3]): in order to obtain an API suitable for Noise as it is, one would define *vr* so that $\mathit{vr}(\mathit{ctx}) = \varepsilon$ for all *ctx*. As this function is not collision resistant, Theorem 6 cannot be used to argue that Noise is secure in the presence of key reuse.

However, this does not imply that any Noise protocol is vulnerable to a cross protocol attack. Indeed, the processing rules provide a defense by binding the handshake pattern and parameters to the state (see [120, Sec. 5.2]) using the hash of the protocol name (e.g., Noise_NK_25519_AESGCM_SHA256) as the initial *HKDF* salt. For each pattern, Noise specifies the initial salt $salt_0$ and a sequence of IKMs $ikm_1, \ldots, ikm_c$, each of which encodes an element of $\mathcal{G}$ or a pre-shared key (PSK). The number of IKMs, and the type of each, depends on the pattern. The i-th call to HKDF has the form $HKDF(salt_{i-1}, ikm_i, \varepsilon)$, where $salt_{i-1}$ was derived from the output of the previous call. By chaining together evaluations of *HKDF* in this way, the sequence of salts computed during the protocol's execution are all bound to the protocol name. Intuitively, this mechanism ought to provide some measure of defense against exposed interface attacks.

Unfortunately, this cannot be proven in our framework. In the first place, it is not clear how to define context separability, since the set of API queries made by different Noise protocols are non-disjoint. In particular, while the first query in each protocol is different—each protocol uses a different initial salt—there is a small chance that subsequent queries might collide (i.e., use the same salt). Moreover, the set of API queries made by the game is not efficiently computable because it depends on the game's random choices. (In contrast, the set $\mathcal{C}_\alpha$ in Eq. (5.14) is efficiently computable, since there is a simple way to check that a given query was made by the game: just check that $info = vr(\alpha)$.)

If we make no restrictions on API queries, or if we merely ensure that the initial salt is distinct, then there is a simple way to differentiate $X/HKDF$ from $\tilde{X}/HKDF$. Choose at random an integer $r \in \mathbb{Z}_{|\mathcal{G}|}$ and ask $L \leftarrow \mathcal{W}(ctx, salt, g^r)$ and $L' \leftarrow \mathcal{W}'(ctx, salt, g^r)$ for some $ctx, salt \in \{0,1\}^*$, where $\mathcal{W}$ (resp. $\mathcal{W}'$) is the name of the main (resp. aux.) interface. In the real experiment, we will have that $L = L'$; but in the reference experiment, we will have $L \neq L'$ with high probability, since the simulator has no information about the adversary's main-interface query. While this differentiator does not yield a cross protocol attack, it does mean we cannot rule these out.

Our analysis leaves open the security of key reuse in Noise as it is. We have shown, however, that a simple tweak of the processing rules (i.e., using the protocol name as the info string for *HKDF*) would close this gap.

Remark 10. PS19 similarly analyze a variant of Noise in which the DH-then-HKDF operation uses the information string as shown in Figure 5.4. But their result (cf. [117, Theorem 7]) is less general than ours, in the sense that their formal model required them to exclude a number of handshake patterns from the analysis. This limitation is a consequence of the strength of their GAP1 security notion (cf. Remark 7): whereas we have found DH-then-HKDF operation to be context separable, this could not be shown in their setting (cf. [117, Theorem 4]). □

5.4 Discussion

Context separability is a property of cryptographic APIs that allows one to "lift" the known security of a given application to security in the presence key reuse. As case studies we examined the APIs of the IETF standard for EdDSA and the Noise protocol framework. The contextualized variants of EdDSA are easily seen to be context separable (Theorem 4); Noise presents some difficulties, but which can be overcome with a slight tweak of the processing rules (Theorem 6). These results suggest that designing APIs to be context separable is not difficult, but provides substantial analytical benefit. In particular, it provides a convenient way to reason about cross protocol attacks that arise as a result of key reuse.

Another application of context separability is to the problem of accounting for extensibility in cryptographic protocols. Two security considerations arise when designing an extension: the first, of course, is that the extension meets its intended security goal; but it is equally important to ensure that extension's availability does not weaken the security goal of the base protocol. As a motivating example, recall the cross protocol attack against TLS 1.2 [126] of Mavrogiannopoulos et al. [105], first discussed in §1.4. Their key-recovery attack results from extending the protocol spec by the addition of ECDH ciphersuites with custom parameters. It exploits the ambiguous encoding of (EC)DH parameters of the ServerKeyExchange message, as well as the lack of binding of the selected ciphersuite (DH or ECDH) to the signature generated by the server. This weakness can be formulated as an exposed interface attack against TLS 1.2, where the API is used by the server to sign messages. Here, the "target application" is the usage of the DH ciphersuite, while the "other applications" (formalized by the exposed interface attack) includes usage of the ECDH ciphersuite.[3]

[3] Note that the term "API" in this attack may not be a system to which the adversary has direct access, but only indirect access by interacting with the server. In particular, "API" refers to the signing interface in the TLS libraries that were vulnerable to the attack.

Viewing their attack in this light, consider how it might have been prevented. Mavrogiannopoulos et al. observe (cf. [105, §6]) that had the server's signature provided an explicit binding of the ciphersuite to the key-exchange parameters, then there would have been no chance of the client misinterpreting ECDH parameters as DH parameters. This is precisely what our notion of context separability aims to formalize: that binding secret key operations to the context in which they are used prevents operations performed in the context of one protocol—or ciphersuite, in this case—from being used to attack another.

Fortunately, this observation appears to be reflected in TLS 1.3 [123]. There the client and server sign all protocol messages, rather than only specific values. (In 1.2, only the key exchange parameters, ClientHello.random, and ServerHello.random are signed.) Because the parameters of the handshake are uniquely determined by this sequence of messages, this provides a cryptographic binding of the handshake to the set of extensions that will be used. An important question, which this dissertation will leave open, is whether this binding could be used to argue that the signing API in TLS 1.3 is context separable. If so, this would provide a simple way to reason about cross protocol attacks among different sets of extensions.

Answering this question requires overcoming at least two interesting challenges. First, the client and server negotiate the signature schemes they use to sign the handshake, just as they do for all other parameters. This means that proving context separability will require us to (1) appeal to the security properties of all supported schemes and (2) address any agility issues [5] that might arise as a result of shared usage of a secret key by two or more schemes. The second issue is that not all extensions are negotiated in the clear. Recall from §4.2 that the server's extension response might appear in the ServerHello, in which case it is transmitted in the clear; but it can also appear later in the sequence of messages (i.e., in EncryptedExtensions...CertificateVerify), in which case the response is encrypted. Consequentially, the selected set of extensions (i.e., the signature's "context") might not be efficiently computable. Despite these challenges, we are optimistic that this methodology would be an effective tool for guiding the design and security analysis of new extensions for TLS.

Chapter 6

Conclusion

While provable security has shaped modern cryptography as we know it, there are a number of ways it might fail to deliver on its promise of "bullet-proof" security [64, 90, 37]. In particular, it succeeds for a given system only to the extent that the formal specification captures the system's real-world behavior: when a discrepancy arises, it is often left to practitioners to interpret its impact on existing analysis. This dissertation offers principled guidance for carrying out this task.

6.1 Summary

When considering a change to a system, the immediate task is to determine whether the translation creates an exploitable weakness. Our main contribution is a formal framework (presented in Chapter 2) in which security of the translated system is proven via a black-box reduction from the security property established by existing analysis. By the "lifting" lemma (Lemma 1) and a "preservation" lemma (Lemma 2 or 4), this reduction is possible whenever the translated system is shared-resource indifferentiable from the original. This approach significantly decreases the analytical effort required to evaluate the security impact of translation: rather than generate a fresh proof for the translated system, one merely needs to reason about the translation itself.

We explored two applications of this methodology. In Chapter 4, we considered the problem of translating a system by revising the scheme's specification. This form of translation is especially common: examples include the translation of a protocol proposed in the literature into a cryptographic standard (e.g., the translation of OPTLS into an early draft of TLS 1.3 [93]); revision of an existing standard by including a new feature (e.g., adding a new ciphersuite to TLS 1.2 [105]); or implementation of an obscure feature not considered in existing analysis (e.g., renegotiation in TLS 1.2 [78]). In order to enable a formal treatment of these kinds of discrepancies, we specified an execution environment for AKE protocols suitable for expressing a variety of security goals in the literature. As a demonstration, we designed an extension for TLS 1.3 that integrates SPAKE2 [4] into the handshake and proved that the execution of the extension is indifferentiable from the execution of SPAKE2. This allowed us to prove the extension is at least as secure as SPAKE2 itself, modulo a modest loss in concrete security (Theorem 3).

In Chapter 5, we considered the problem of translating a system by changing the scheme's execution environment. An important example, which we addressed in this work, is translation by reuse of a system's secret keys. We formalized an attack model for this setting that considers the security of an application of interest when access to the underlying API is also shared by other applications. Within the translation framework we formulated a property of the API, called context separability, that allows one to prove security in the presence of key reuse by appealing to the application's existing analysis. Context separability reflects a design pattern already apparent in a variety of cryptographic standards, two of which we studied in detail [87, 120] (Theorems 4 and 6).

Taken together, the case studies considered in this work represent only a small fraction of the discrepancies that might arise between a real system and its formal specification. In light of this gap, techniques have advanced considerably in recent years for rigorously modeling real-world cryptography in all of its complexity. (We exhibited one such technique in Chapter 3.) But because these systems are subject to continual change, the task of ensuring that their security rests on firm, formal foundations remains an on-going challenge. Our hope is that this dissertation will help guide this process.

Limitations. Although the translation framework is useful for evaluating the impact of changes to a cryptosystem, we emphasize that it is not always the best tool for the task at hand. First, when the security goal of the translated protocol differs from that of the original, our framework (in its current form) does not provide a way to argue that the new goal is achieved: in this case, a fresh proof is still required. Second, the real system might be so different from the reference system that an indifferentiability-style argument is impossible (cf. Theorem 5). Our methodology is conservative in the sense that indifferentiability is sufficient, but not necessary, to prove the desired claim: ruling out indifferentiability does not necessarily imply a concrete attack against the real system.

6.2 Future Work

We expect the translation framework to have high value for the development of cryptographic standards, since it provides a way to quickly vet changes to a protocol before the changes are implemented—and without the need for expert knowledge of the state of the system's proven security. We believe there are a number of standardization efforts underway that stand to benefit from such a tool.

- We have already mentioned in Chapter 4 the PAKE-standardization effort of the CFRG. Each of the schemes being considered is based on a protocol originally proposed in the academic literature (e.g., [4, 86, 81]): proving indifferentiability of the standard from the original scheme ensures that whatever security supports the latter also applies to the standard.
- There are a number of extensions for TLS being considered, including PAKE extensions [143, 16], other modes of authentication [17, 128], and new features that change the protocol's security properties [127, 119]. Each of these extensions requires supporting analysis, both to establish the security of the extension itself and to ensure that the extension's availability does not create a vulnerability. (See the discussion at the end of Chapter 5 for one possible approach.)
- An effort currently in its infancy is the so-called "Compact TLS (cTLS)" protocol [125], which aims to provide the same level of security of TLS 1.3, but without the bloat that resulted from the need to maintain backwards compatibility with previous versions of the protocol. Ideally, the security of cTLS would follow from the security of TLS 1.3.
- Similarly, the key-exchange protocol in the most recent iteration of QUIC [84] is based on the TLS 1.3 handshake; but QUIC is designed to run in an execution environment in which packets are routinely dropped or delivered out-of-order. (In particular, QUIC does not assume reliable transport, like TLS does.) An interesting question is whether the handshake's security is preserved in this new execution environment.

Our framework is well-suited to support many of these efforts. However, given the "pen-and-paper" nature of the security proofs presented in this dissertation, it remains to be seen whether our approach scales to more complex analyses or more sophisticated systems (e.g., the Messaging Layer Security (MLS) protocol [15]).

Indeed, the rigor of code-based game-playing proofs is limited, ultimately, by the (in)formality of its pseudocode [33]. This is also true of the pseudocode in this dissertation: while we have endeavored to ensure that the semantics of worlds, adversaries, simulators, and so on is clear in context, we have not attempted to fully specify objects' syntax or formalize their semantics. Doing so—or perhaps reformulating our framework in an existing language with a suitable formal semantics—could ease the task of proving (and verifying proofs of) secure translation. Whether mechanization of proofs in the translation framework is possible remains an open question since, to the best of our knowledge, no one has attempted a machine-checked proof of indifferentiability. However, concurrent work has yielded promising results: a proof of security in the UC framework [53] has recently been demonstrated for the EasyCrypt proof system [58]; and given the similarities between UC and indifferentiability (cf. [104, §3.3]), this suggests that mechanization of indifferentiability arguments might be feasible.

Bibliography

[1] Abdalla, M., Barbosa, M.: Perfect forward security of SPAKE2. Cryptology ePrint Archive, Report 2019/1194 (2019), `https://eprint.iacr.org/2019/1194`

[2] Abdalla, M., Bellare, M., Rogaway, P.: The oracle Diffie-Hellman assumptions and an analysis of DHIES. In: Topics in Cryptology — CT-RSA 2001. pp. 143–158. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)

[3] Abdalla, M., Fouque, P.A., Pointcheval, D.: Password-based authenticated key exchange in the three-party setting. In: Vaudenay, S. (ed.) Public Key Cryptography — PKC 2005. pp. 65–84. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)

[4] Abdalla, M., Pointcheval, D.: Simple password-based encrypted key exchange protocols. In: Menezes, A. (ed.) Topics in Cryptology — CT-RSA 2005. pp. 191–208. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)

[5] Acar, T., Belenkiy, M., Bellare, M., Cash, D.: Cryptographic agility and its relation to circular encryption. In: Advances in Cryptology — EUROCRYPT 2010. pp. 403–422. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)

[6] Acar, T., Nguyen, L., Zaverucha, G.: A TPM Diffie-Hellman oracle. Cryptology ePrint Archive, Report 2013/667 (2013), `https://eprint.iacr.org/2013/667`

[7] Acar, Y., Backes, M., Fahl, S., Garfinkel, S., Kim, D., Mazurek, M.L., Stransky, C.: Comparing the usability of cryptographic APIs. In: 2017 IEEE Symposium on Security and Privacy. pp. 154–171 (May 2017)

[8] Albrecht, M.R., Degabriele, J.P., Hansen, T.B., Paterson, K.G.: A surfeit of SSH cipher suites. In: Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security. pp. 1480–1491. ACM (2016)

[9] Albrecht, M.R., Paterson, K.G., Watson, G.J.: Plaintext recovery attacks against SSH. In: Proceedings of the 30th IEEE Symposium on Security and Privacy. pp. 16–26. IEEE (2009)

[10] AlFardan, N.J., Paterson, K.G.: Lucky Thirteen: Breaking the TLS and DTLS record protocols. In: 2013 IEEE Symposium on Security and Privacy. pp. 526–540. IEEE (2013)

[11] Allen, C., Dierks, T.: The TLS protocol version 1.0. RFC 2246 (Jan 1999), `https://rfc-editor.org/rfc/rfc2246.txt`

[12] Andreeva, E., Bogdanov, A., Luykx, A., Mennink, B., Mouha, N., Yasuda, K.: How to securely release unverified plaintext in authenticated encryption. In: Advances in Cryptology — ASIACRYPT 2014. pp. 105–125. Springer Berlin Heidelberg (2014)

[13] Badertscher, C., Matt, C., Maurer, U., Rogaway, P., Tackmann, B.: Augmented secure channels and the goal of the TLS 1.3 record layer. In: Provable Security. pp. 85–104. Springer International Publishing (2015)

[14] Barbosa, M., Farshim, P.: Indifferentiable authenticated encryption. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology — CRYPTO 2018. pp. 187–220. Springer International Publishing, Cham (2018)

[15] Barnes, R., Beurdouche, B., Millican, J., Omara, E., Cohn-Gordon, K., Robert, R.: The messaging layer security (MLS) protocol. Internet-Draft draft-ietf-mls-protocol-09, Internet Engineering Task Force (Mar 2020), `https://datatracker.ietf.org/doc/html/draft-ietf-mls-protocol-09`

[16] Barnes, R., Friel, O.: Usage of PAKE with TLS 1.3. Internet-Draft draft-barnes-tls-pake-04, Internet Engineering Task Force (Jul 2018), `https://datatracker.ietf.org/doc/html/draft-barnes-tls-pake-04`

[17] Barnes, R., Iyengar, S., Sullivan, N., Rescorla, E.: Delegated credentials for TLS. Internet-Draft draft-ietf-tls-subcerts-07, Internet Engineering Task Force (Mar 2020), `https://datatracker.ietf.org/doc/html/draft-ietf-tls-subcerts-07`

[18] Barwell, G., Page, D., Stam, M.: Rogue decryption failures: Reconciling AE robustness notions. In: Proceedings of the 15th IMA International Conference on Cryptography and Coding. pp. 94–111. Springer International Publishing (2015)

[19] Basin, D., Cremers, C., Dreier, J., Sasse, R.: Symbolically analyzing security protocols using Tamarin. ACM SIGLOG News **4**(4), 19–30 (Nov 2017), `http://doi.acm.org/10.1145/3157831.3157835`

[20] Becerra, J., Ostrev, D., Skrobot, M.: Forward secrecy of SPAKE2. In: Baek, J., Susilo, W., Kim, J. (eds.) Provable Security. pp. 366–384. Springer International Publishing, Cham (2018)

[21] Bellare, M., Desai, A., Jokipii, E., Rogaway, P.: A concrete security treatment of symmetric encryption. In: Proceedings 38th Annual Symposium on Foundations of Computer Science. pp. 394–403 (Oct 1997), `https://doi.org/10.1109/SFCS.1997.646128`

[22] Bellare, M., Davis, H., Günther, F.: Separate your domains: NIST PQC KEMs, oracle cloning and read-only indifferentiability. Cryptology ePrint Archive, Report 2020/241 (2020), `https://eprint.iacr.org/2020/241`

[23] Bellare, M., Keelveedhi, S., Ristenpart, T.: Message-locked encryption and secure deduplication. In: Johansson, T., Nguyen, P.Q. (eds.) Advances in Cryptology — EUROCRYPT 2013. pp. 296–312. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)

[24] Bellare, M., Kohno, T., Namprempre, C.: Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the Encode-then-Encrypt-and-MAC paradigm. ACM Trans. Inf. Syst. Secur. **7**(2), 206–241 (2004)

[25] Bellare, M., Meiklejohn, S., Thomson, S.: Key-versatile signatures and applications: RKA, KDM and joint Enc/Sig. In: Advances in Cryptology — EUROCRYPT 2014. pp. 496–513. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)

[26] Bellare, M., Namprempre, C.: Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. Cryptology ePrint Archive, Report 2000/025 (2000), `https://eprint.iacr.org/2000/025`

[27] Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) Advances in Cryptology — EUROCRYPT 2000. pp. 139–155. Springer Berlin Heidelberg, Berlin, Heidelberg (2000)

[28] Bellare, M., Ristenpart, T., Tessaro, S.: Multi-instance security and its application to password-based cryptography. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology — CRYPTO 2012. pp. 312–329. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)

[29] Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Proceedings of the 1st ACM Conference on Computer and Communications Security. pp. 62–73. CCS '93, ACM, New York, NY, USA (1993)

[30] Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Advances in Cryptology — CRYPTO '93. pp. 232–249. Springer Berlin Heidelberg, Berlin, Heidelberg (1994)

[31] Bellare, M., Rogaway, P.: Provably secure session key distribution: The three party case. In: Proceedings of the Twenty-seventh Annual ACM Symposium on Theory of Computing. pp. 57–66. STOC '95, ACM, New York, NY, USA (1995), `http://doi.acm.org/10.1145/225058.225084`

[32] Bellare, M., Rogaway, P.: The exact security of digital signatures: How to sign with RSA and Rabin. In: Maurer, U. (ed.) Advances in Cryptology — EUROCRYPT '96. pp. 399–416. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)

[33] Bellare, M., Rogaway, P.: Code-based game-playing proofs and the security of triple encryption. Cryptology ePrint Archive, Report 2004/331 (2004), `https://eprint.iacr.org/2004/331`

[34] Bellare, M., Tackmann, B.: The multi-user security of authenticated encryption: AES-GCM in TLS 1.3. In: Advances in Cryptology — CRYPTO 2016. pp. 247–276. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)

[35] Bellovin, S.M., Merritt, M.: Encrypted key exchange: Password-based protocols secure against dictionary attacks. In: Proceedings 1992 IEEE Computer Society Symposium on Research in Security and Privacy. pp. 72–84 (May 1992), `https://doi.org/10.1109/RISP.1992.213269`

[36] Bergsma, F., Dowling, B., Kohlar, F., Schwenk, J., Stebila, D.: Multi-ciphersuite security of the Secure Shell (SSH) protocol. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. pp. 369–381. CCS '14, ACM, New York, NY, USA (2014), `http://doi.acm.org/10.1145/2660267.2660286`

[37] Bernstein, D.J.: Comparing proofs of security for lattice-based encryption. Cryptology ePrint Archive, Report 2019/691 (2019), `https://eprint.iacr.org/2019/691`

[38] Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.Y.: High-speed high-security signatures. Journal of Cryptographic Engineering **2**(2), 77–89 (Sep 2012)

[39] Bhargavan, K., Blanchet, B., Kobeissi, N.: Verified models and reference implementations for the TLS 1.3 standard candidate. In: 2017 IEEE Symposium on Security and Privacy. pp. 483–502 (May 2017), `https://doi.org/10.1109/SP.2017.26`

[40] Bhargavan, K., Boureanu, I., Fouque, P., Onete, C., Richard, B.: Content delivery over TLS: a cryptographic analysis of Keyless SSL. In: 2017 IEEE European Symposium on Security and Privacy. pp. 1–16 (April 2017), `https://doi.org/10.1109/EuroSP.2017.52`

[41] Bhargavan, K., Brzuska, C., Fournet, C., Green, M., Kohlweiss, M., Zanella-Béguelin, S.: Downgrade resilience in key-exchange protocols. In: 2016 IEEE Symposium on Security and Privacy. pp. 506–525 (May 2016), `https://doi.org/10.1109/SP.2016.37`

[42] Bhargavan, K., Fournet, C., Kohlweiss, M., Pironti, A., Strub, P.: Implementing TLS with verified cryptographic security. In: 2013 IEEE Symposium on Security and Privacy. pp. 445–459 (May 2013), `https://doi.org/10.1109/SP.2013.37`

[43] Bhargavan, K., Lavaud, A.D., Fournet, C., Pironti, A., Strub, P.Y.: Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS. In: 2014 IEEE Symposium on Security and Privacy. pp. 98–113 (May 2014), `https://doi.org/10.1109/SP.2014.14`

[44] Bhargavan, K.: Review of the balanced PAKE proposals. Mail to IRTF CFRG, September 2019 (2019), `https://mailarchive.ietf.org/arch/msg/cfrg/5VhZLYGpzU8MWPlbMr2cf4Uc-nI`

[45] Bhargavan, K., Fournet, C., Kohlweiss, M., Pironti, A., Strub, P.Y., Zanella-Béguelin, S.: Proving the TLS handshake secure (as it is). In: Advances in Cryptology — CRYPTO 2014. pp. 235–255. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)

[46] Blanchet, B.: A computationally sound mechanized prover for security protocols. IEEE Transactions on Dependable and Secure Computing **5**(4), 193–207 (Oct 2008), `https://doi.org/10.1109/TDSC.2007.1005`

[47] Blanchet, B.: Modeling and verifying security protocols with the Applied Pi Calculus and ProVerif. Found. Trends Priv. Secur. **1**(1-2), 1–135 (Oct 2016), `https://doi.org/10.1561/3300000004`

[48] Boldyreva, A., Degabriele, J.P., Paterson, K.G., Stam, M.: Security of symmetric encryption in the presence of ciphertext fragmentation. In: Pointcheval, D., Johansson, T. (eds.) Advances in Cryptology — EUROCRYPT 2012. pp. 682–699. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)

[49] Boldyreva, A., Degabriele, J.P., Paterson, K.G., Stam, M.: On symmetric encryption with distinguishable decryption failures. In: Fast Software Encryption. pp. 367–390. Springer Berlin Heidelberg (2014)

[50] Brickell, E., Camenisch, J., Chen, L.: Direct anonymous attestation. In: Proceedings of the 11th ACM Conference on Computer and Communications Security. pp. 132–145. CCS '04, ACM, New York, NY, USA (2004)

[51] Brown, D.R.L., Gallant, R.P.: The static Diffie-Hellman problem. Cryptology ePrint Archive, Report 2004/306 (2004), `https://eprint.iacr.org/2004/306`

[52] Camenisch, J., Chen, L., Drijvers, M., Lehmann, A., Novick, D., Urian, R.: One TPM to bind them all: Fixing TPM 2.0 for provably secure anonymous attestation. In: 2017 IEEE Symposium on Security and Privacy. pp. 901–920 (May 2017)

[53] Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067 (2000), https://eprint.iacr.org/2000/067

[54] Canetti, R., Dodis, Y., Pass, R., Walfish, S.: Universally composable security with global setup. In: Vadhan, S.P. (ed.) Theory of Cryptography. pp. 61–85. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)

[55] Canetti, R., Fischlin, M.: Universally composable commitments. Cryptology ePrint Archive, Report 2001/055 (2001), https://eprint.iacr.org/2001/055

[56] Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.: Universally composable password-based key exchange. In: Cramer, R. (ed.) Advances in Cryptology — EUROCRYPT 2005. pp. 404–421. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)

[57] Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Advances in Cryptology — EUROCRYPT 2001. pp. 453–474. Springer Berlin Heidelberg (2001)

[58] Canetti, R., Stoughton, A., Varia, M.: Easyuc: Using EasyCrypt to mechanize proofs of universally composable security. Cryptology ePrint Archive, Report 2019/582 (2019), https://eprint.iacr.org/2019/582

[59] Cash, D., Kiltz, E., Shoup, V.: The twin Diffie-Hellman problem and applications. In: Smart, N. (ed.) Advances in Cryptology — EUROCRYPT 2008. pp. 127–145. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)

[60] Coron, J.S., Dodis, Y., Malinaud, C., Puniya, P.: Merkle-Damgård revisited: How to construct a hash function. In: Advances in Cryptology — CRYPTO 2005. pp. 430–448. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)

[61] Coron, J.S., Holenstein, T., Künzler, R., Patarin, J., Seurin, Y., Tessaro, S.: How to build an ideal cipher: The indifferentiability of the Feistel construction. Journal of Cryptology **29**(1), 61–114 (Jan 2016), https://doi.org/10.1007/s00145-014-9189-6

[62] Cremers, C., Feltz, M.: Beyond eCK: Perfect forward secrecy under actor compromise and ephemeral-key reveal. Cryptology ePrint Archive, Report 2012/416 (2012), https://eprint.iacr.org/2012/416

[63] Csiszár, I., Körner, J.: Information Theory: Coding Theorems for Discrete Memoryless Systems. Cambridge University Press (2011)

[64] Degabriele, J.P., Paterson, K., Watson, G.: Provable security in the real world. IEEE Security & Privacy **9**(3), 33–41 (2011)

[65] Degabriele, J.P., Lehmann, A., Paterson, K.G., Smart, N.P., Strefler, M.: On the joint security of encryption and signature in EMV. In: Topics in Cryptology — CT-RSA 2012. pp. 116–135. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)

[66] Degabriele, J.P., Paterson, K.G.: On the (in)security of IPsec in MAC-then-Encrypt configurations. In: Proceedings of the 17th ACM Conference on Computer and Communications Security. pp. 493–504. ACM (2010)

[67] Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Protzenko, J., Rastogi, A., Swamy, N., Zanella-Beguelin, S., Bhargavan, K., Pan, J., Zinzindohoue, J.K.: Implementing and proving the TLS 1.3 record layer. In: 2017 IEEE Symposium on Security and Privacy. pp. 463–482 (May 2017), https://doi.org/10.1109/SP.2017.58

[68] Diffie, W., Hellman, M.: New directions in cryptography. IEEE Trans. Inf. Theor. **22**(6), 644–654 (Sep 2006), http://dx.doi.org/10.1109/TIT.1976.1055638

[69] Dolev, D., Yao, A.C.: On the security of public key protocols. In: Proceedings of the 22nd Annual Symposium on Foundations of Computer Science. pp. 350–357. SFCS '81, IEEE Computer Society, Washington, DC, USA (1981), https://doi.org/10.1109/SFCS.1981.32

[70] Donenfeld, J.A.: WireGuard: Next generation kernel network tunnel. Online white paper (June 2018), https://www.wireguard.com/papers/wireguard.pdf

[71] Dowling, B., Paterson, K.G.: A cryptographic analysis of the WireGuard protocol. In: Preneel, B., Vercauteren, F. (eds.) Applied Cryptography and Network Security. pp. 3–21. Springer International Publishing, Cham (2018)

[72] EMVCo: EMV integrated circuit card specifications for payment systems (November 2011), https://www.emvco.com/emv-technologies/contact/

[73] Faz-Hernandez, A., Scott, S., Sullivan, N., Wahby, R.S., Wood, C.A.: Hashing to elliptic curves. Internet-Draft draft-irtf-cfrg-hash-to-curve-07, Internet Engineering Task Force (Apr 2020), https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-hash-to-curve-07

[74] Fischlin, M., Günther, F., Schmidt, B., Warinschi, B.: Key confirmation in key exchange: A formal treatment and implications for TLS 1.3. In: 2016 IEEE Symposium on Security and Privacy. pp. 452–469 (May 2016), https://doi.org/10.1109/SP.2016.34

[75] Fischlin, M., Günther, F., Marson, G.A., Paterson, K.G.: Data is a stream: Security of stream-based channels. In: Advances in Cryptology — CRYPTO 2015. pp. 545–564. Springer Berlin Heidelberg (2015)

[76] Fischlin, M., Günther, F., Marson, G.A., Paterson, K.G.: Data is a stream: Security of stream-based channels. Cryptology ePrint Archive, Report 2017/1191 (2017), https://eprint.iacr.org/2017/1191

[77] Fischlin, M., Lehmann, A., Ristenpart, T., Shrimpton, T., Stam, M., Tessaro, S.: Random oracles with(out) programmability. In: Abe, M. (ed.) Advances in Cryptology - ASIACRYPT 2010. pp. 303–320. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)

[78] Giesen, F., Kohlar, F., Stebila, D.: On the security of TLS renegotiation. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security. pp. 387–398. CCS '13, ACM, New York, NY, USA (2013), http://doi.acm.org/10.1145/2508859.2516694

[79] Google: QUIC, a multiplexed stream transport over UDP, https://www.chromium.org/quic, accessed 13 Feb 2018

[80] Green, M., Droms, R., Housley, R., Turner, P., Fenter, S.: Data center use of static Diffie-Hellman in TLS 1.3. Internet-Draft draft-green-tls-static-dh-in-tls13-01, Internet Engineering Task Force (Jul 2017), https://datatracker.ietf.org/doc/html/draft-green-tls-static-dh-in-tls13-01

[81] Haase, B., Labrique, B.: AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. Cryptology ePrint Archive, Report 2018/286 (2018), https://eprint.iacr.org/2018/286

[82] Haber, S., Pinkas, B.: Securely combining public-key cryptosystems. In: Proceedings of the 8th ACM Conference on Computer and Communications Security. pp. 215–224. CCS '01, ACM, New York, NY, USA (2001)

[83] Hoang, V.T., Krovetz, T., Rogaway, P.: Robust authenticated-encryption: AEZ and the problem that it solves. In: Advances in Cryptology — EUROCRYPT 2015. pp. 15–44. Springer Berlin Heidelberg (2015)

[84] Iyengar, J., Thomson, M.: QUIC: a UDP-based multiplexed and secure transport. Internet-Draft draft-ietf-quic-transport-27, Internet Engineering Task Force (Feb 2020), https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-27

[85] Jager, T., Kohlar, F., Schäge, S., Schwenk, J.: On the security of TLS-DHE in the standard model. In: Advances in Cryptology — CRYPTO 2012. pp. 273–293. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)

[86] Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. Cryptology ePrint Archive, Report 2018/163 (2018), https://eprint.iacr.org/2018/163

[87] Josefsson, S., Liusvaara, I.: Edwards-curve digital signature algorithm (EdDSA). RFC 8032 (Jan 2017), https://rfc-editor.org/rfc/rfc8032.txt

[88] Kelsey, J., Schneier, B., Wagner, D.: Protocol interactions and the chosen protocol attack. In: Security Protocols. pp. 91–104. Springer Berlin Heidelberg, Berlin, Heidelberg (1998)

[89] Kobeissi, N., Nicolas, G., Bhargavan, K.: Noise Explorer: Fully automated modeling and verification for arbitrary Noise protocols. In: 2019 IEEE European Symposium on Security and Privacy. pp. 356–370 (June 2019), https://doi.org/10.1109/EuroSP.2019.00034

[90] Koblitz, N., Menezes, A.: Critical perspectives on provable security: Fifteen years of "another look" papers. Cryptology ePrint Archive, Report 2019/1336 (2019), https://eprint.iacr.org/2019/1336

[91] Kohlar, F., Schäge, S., Schwenk, J.: On the security of TLS-DH and TLS-RSA in the standard model. Cryptology ePrint Archive, Report 2013/367 (2013), https://eprint.iacr.org/2013/367

[92] Krawczyk, D.H., Eronen, P.: HMAC-based Extract-and-Expand key derivation function (HKDF). RFC 5869 (May 2010), https://rfc-editor.org/rfc/rfc5869.txt

[93] Krawczyk, H., Wee, H.: The OPTLS protocol and TLS 1.3. In: 2016 IEEE European Symposium on Security and Privacy. pp. 81–96 (March 2016), https://doi.org/10.1109/EuroSP.2016.18

[94] Krawczyk, H.: SIGMA: The 'SIGn-and-MAc' approach to authenticated Diffie-Hellman and its use in the IKE protocols. In: Boneh, D. (ed.) Advances in Cryptology - CRYPTO 2003. pp. 400–425. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)

[95] Krawczyk, H.: HMQV: A high-performance secure Diffie-Hellman protocol. Cryptology ePrint Archive, Report 2005/176 (2005), https://eprint.iacr.org/2005/176

[96] Krawczyk, H., Paterson, K.G., Wee, H.: On the security of the TLS protocol: A systematic analysis. In: Advances in Cryptology — CRYPTO 2013. pp. 429–448. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)

[97] Künnemann, R., Steel, G.: YubiSecure? Formal security analysis results for the Yubikey and YubiHSM. In: Security and Trust Management. pp. 257–272. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)

[98] Ladd, W., Kaduk, B.: SPAKE2, a PAKE. Internet-Draft draft-irtf-cfrg-spake2-10, Internet Engineering Task Force (Feb 2020), https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-spake2-10

[99] LaMacchia, B., Lauter, K., Mityagin, A.: Stronger security of authenticated key exchange. In: Susilo, W., Liu, J.K., Mu, Y. (eds.) Provable Security. pp. 1–16. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)

[100] Li, X., Xu, J., Zhang, Z., Feng, D., Hu, H.: Multiple handshakes security of TLS 1.3 candidates. In: 2016 IEEE Symposium on Security and Privacy. pp. 486–505 (May 2016), https://doi.org/10.1109/SP.2016.36

[101] Lim, C.H., Lee, P.J.: A key recovery attack on discrete log-based schemes using a prime order subgroup. In: Advances in Cryptology — CRYPTO '97. pp. 249–263. Springer Berlin Heidelberg, Berlin, Heidelberg (1997)

[102] Lipp, B., Blanchet, B., Bhargavan, K.: A mechanised cryptographic proof of the WireGuard virtual private network protocol. In: 2019 IEEE European Symposium on Security and Privacy. pp. 231–246 (June 2019), https://doi.org/10.1109/EuroSP.2019.00026

[103] Lonvick, C.M., Ylonen, T.: The Secure Shell (SSH) protocol architecture. RFC 4251 (Jan 2006). https://doi.org/10.17487/RFC4251, https://rfc-editor.org/rfc/rfc4251.txt

[104] Maurer, U., Renner, R., Holenstein, C.: Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In: Theory of Cryptography. pp. 21–39. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)

[105] Mavrogiannopoulos, N., Vercauteren, F., Velichkov, V., Preneel, B.: A cross-protocol attack on the TLS protocol. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security. pp. 62–72. CCS '12, ACM, New York, NY, USA (2012), http://doi.acm.org/10.1145/2382196.2382206

[106] McGrew, D.: An interface and algorithms for authenticated encryption. RFC 5116 (Jan 2008), https://rfc-editor.org/rfc/rfc5116.txt

[107] Moeller, B., Bolyard, N., Gupta, V., Blake-Wilson, S., Hawk, C.: Elliptic curve cryptography (ECC) cipher suites for Transport Layer Security (TLS). RFC 4492 (May 2006), https://rfc-editor.org/rfc/rfc4492.txt

[108] Namprempre, C., Rogaway, P., Shrimpton, T.: Reconsidering generic composition. In: Advances in Cryptology — EUROCRYPT 2014. pp. 257–274. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)

[109] Okamoto, T., Pointcheval, D.: The gap-problems: A new class of problems for the security of cryptographic schemes. In: Public Key Cryptography. pp. 104–118. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)

[110] Oliveira, D., Rosenthal, M., Morin, N., Yeh, K.C., Cappos, J., Zhuang, Y.: It's the psychology stupid: How heuristics explain software vulnerabilities and how priming can illuminate developer's blind spots. In: Proceedings of the 30th Annual Computer Security Applications Conference. pp. 296–305. ACSAC '14, ACM, New York, NY, USA (2014)

[111] Oliveira, D.S., Lin, T., Rahman, M.S., Akefirad, R., Ellis, D., Perez, E., Bobhate, R., DeLong, L.A., Cappos, J., Brun, Y.: API blindspots: Why experienced developers write vulnerable code. In: Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018). pp. 315–328. USENIX Association, Baltimore, MD (2018)

[112] Paterson, K.G., AlFardan, N.J.: Plaintext-recovery attacks against datagram TLS. In: 19th Annual Network and Distributed System Security Symposium, NDSS (2012)

[113] Paterson, K.G., van der Merwe, T.: Reactive and proactive standardisation of TLS. In: Security Standardisation Research. pp. 160–186. Springer International Publishing, Cham (2016)

[114] Paterson, K.G., Ristenpart, T., Shrimpton, T.: Tag size *does* matter: Attacks and proofs for the TLS record protocol. In: Advances in Cryptology — ASIACRYPT 2011. pp. 372–389. Springer Berlin Heidelberg (2011)

[115] Paterson, K.G., Schuldt, J.C.N., Stam, M., Thomson, S.: On the joint security of encryption and signature, revisited. In: Advances in Cryptology — ASIACRYPT 2011. pp. 161–178. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)

[116] Patton, C., Shrimpton, T.: Partially specified channels: The TLS 1.3 record layer without elision. Cryptology ePrint Archive, Report 2018/634 (2018), `https://eprint.iacr.org/2018/634`

[117] Patton, C., Shrimpton, T.: Security in the presence of key reuse: Context-separable interfaces and their applications. Cryptology ePrint Archive, Report 2019/519 (2019), `https://eprint.iacr.org/2019/519`

[118] Patton, C., Shrimpton, T.: Quantifying the security cost of migrating protocols to practice. Cryptology ePrint Archive, Report 2020/573 (2020), `https://eprint.iacr.org/2020/573`

[119] Pauly, T., Schinazi, D., Wood, C.A.: TLS ticket requests. Internet-Draft draft-ietf-tls-ticketrequests-05, Internet Engineering Task Force (Apr 2020), `https://datatracker.ietf.org/doc/html/draft-ietf-tls-ticketrequests-05`

[120] Perrin, T.: The Noise protocol framework. Online white paper (July 2018), `https://noiseprotocol.org/noise.html`

[121] Pollard, J.M.: Kangaroos, monopoly and discrete logarithms. J. Cryptol. **13**(4), 437–447 (2000)

[122] Rescorla, E., Ray, M., Dispensa, S., Oskov, N.: Transport Layer Security (TLS) renegotiation indication extension. RFC 5746, RFC Editor (February 2010), `http://www.rfc-editor.org/rfc/rfc5746.txt`

[123] Rescorla, E.: The Transport Layer Security (TLS) protocol version 1.3. RFC 8446 (Aug 2018), `https://rfc-editor.org/rfc/rfc8446.txt`

[124] Rescorla, E.: The Transport Layer Security (TLS) protocol version 1.3. Internet-Draft draft-ietf-tls-tls13-23, Internet Engineering Task Force (2018), `https://datatracker.ietf.org/doc/html/draft-ietf-tls-tls13-23`

[125] Rescorla, E., Barnes, R., Tschofenig, H.: Compact TLS 1.3. Internet-Draft draft-ietf-tls-ctls-00, Internet Engineering Task Force (Apr 2020), `https://datatracker.ietf.org/doc/html/draft-ietf-tls-ctls-00`

[126] Rescorla, E., Dierks, T.: The Transport Layer Security (TLS) protocol version 1.2. RFC 5246 (Aug 2008), `https://rfc-editor.org/rfc/rfc5246.txt`

[127] Rescorla, E., Oku, K., Sullivan, N., Wood, C.A.: Encrypted server name indication for TLS 1.3. Internet-Draft draft-ietf-tls-esni-06, Internet Engineering Task Force (Mar 2020), `https://datatracker.ietf.org/doc/html/draft-ietf-tls-esni-06`

[128] Rescorla, E., Sullivan, N., Wood, C.A.: Semi-static Diffie-Hellman key establishment for TLS 1.3. Internet-Draft draft-ietf-tls-semistatic-dh-01, Internet Engineering Task Force (Mar 2020), `https://datatracker.ietf.org/doc/html/draft-ietf-tls-semistatic-dh-01`

[129] Ristenpart, T., Shacham, H., Shrimpton, T.: Careful with composition: Limitations of the indifferentiability framework. In: Advances in Cryptology — EUROCRYPT 2011. pp. 487–506. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)

[130] Rogaway, P., Stegers, T.: Authentication without elision: Partially specified protocols, associated data, and cryptographic models described by code. In: 2009 22nd IEEE Computer Security Foundations Symposium. pp. 26–39 (July 2009)

[131] Rogaway, P.: Authenticated-encryption with associated-data. In: Proceedings of the 9th ACM Conference on Computer and Communications Security. pp. 98–107. CCS '02, Association for Computing Machinery, New York, NY, USA (2002), https://doi.org/10.1145/586110.586125

[132] Rogaway, P.: Practice-oriented provable security and the social construction of cryptography. Essay accompanying an invited talk at EUROCRYPT 2009 (2009), http://web.cs.ucdavis.edu/~rogaway/papers/cc.pdf

[133] Rogaway, P., Zhang, Y.: Simplifying game-based definitions. In: Advances in Cryptology — CRYPTO 2018. pp. 3–32. Springer International Publishing, Cham (2018)

[134] Ronen, E., Gillham, R., Genkin, D., Shamir, A., Wong, D., Yarom, Y.: The 9 lives of Bleichenbacher's CAT: New Cache ATtacks on TLS implementations. Cryptology ePrint Archive, Report 2018/1173 (2018), https://eprint.iacr.org/2018/1173

[135] Schnorr, C.P.: Efficient signature generation by smart cards. Journal of Cryptology **4**(3), 161–174 (Jan 1991)

[136] Seo, K., Kent, S.: Security architecture for the Internet protocol. RFC 4301 (Dec 2005), https://rfc-editor.org/rfc/rfc4301.txt

[137] Shoup, V.: Security analysis of SPAKE2+. Cryptology ePrint Archive, Report 2020/313 (2020), https://eprint.iacr.org/2020/313

[138] Skrobot, M., Lancrenon, J.: On composability of game-based password authenticated key exchange. In: 2018 IEEE European Symposium on Security and Privacy. pp. 443–457 (April 2018), https://doi.org/10.1109/EuroSP.2018.00038

[139] Smyshlyaev, S.: Overview of existing PAKEs and PAKE selection criteria. IETF 104 (2019), https://datatracker.ietf.org/meeting/104/materials/slides-104-cfrg-pake-selection

[140] Smyth, B., Pironti, A.: Truncating TLS connections to violate beliefs in web applications. In: 7th USENIX Workshop on Offensive Technologies. USENIX (2013)

[141] Stevens, M., Bursztein, E., Karpman, P., Albertini, A., Markov, Y.: The first collision for full SHA-1. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology — CRYPTO 2017. pp. 570–596. Springer International Publishing, Cham (2017)

[142] Sullivan, N.: Keyless SSL: The nitty gritty technical details, https://blog.cloudflare.com/keyless-ssl-the-nitty-gritty-technical-details/, accessed 16 April 2020

[143] Sullivan, N., Krawczyk, D.H., Friel, O., Barnes, R.: Usage of OPAQUE with TLS 1.3. Internet-Draft draft-sullivan-tls-opaque-00, Internet Engineering Task Force (Mar 2019), https://datatracker.ietf.org/doc/html/draft-sullivan-tls-opaque-00, Work in progress

[144] Tackmann, B.: PAKE review. Mail to IRTF CFRG, October 2019 (2019), https://mailarchive.ietf.org/arch/msg/cfrg/1sNu9USxo1lnFdzCL5msUFKBjzM

[145] Trusted Computing Group: TPM 2.0 library specification (September 2016), https://trustedcomputinggroup.org/resource/tpm-library-specification/

[146] Vaudenay, S.: Security flaws induced by CBC padding — Applications to SSL, IPSEC, WTLS... In: Advances in Cryptology — EUROCRYPT 2002. pp. 534–545. Springer Berlin Heidelberg (2002)